PC Structured Knowledgebases

Hypertext Construction Toolkit

HyperLink™

- Identify ideas
- Build indexes
- Link references
- Create networks
- Convert structures
- Access knowledge
- Verify integrity
- Track changes
- Condense systems

How to rapidly build hypertext systems

Includes PC-Hypertext

[to NELSON
Jan 90
[sent by McEnroe, Cleo's]

HyperLink 89

Neil Larson
MaxThink
44 Rincon Rd
Kensington, CA 94707

Vioce: 415-540-5508
BBS: 415-540-6114

Copyright MaxThink 1989

Table of Contents

		page
Chapter 1	Introduction	1
Chapter 2	Overview	3
Chapter 3	Commands and formats	8
Chapter 4	How to run PC-Hypertext	14
Chapter 5	Setting up shop	16
Chapter 6	Your first network	18
Chapter 7	Adding word processor links	21
Chapter 8	Collecting the knowledge	22
Chapter 9	Creating ASCII nodes	25
Chapter 10	Cross-linking ASCII files	31
Chapter 11	Checking links	38
Chapter 12	Building network links	42
Chapter 13	Distribution issues	45
Appendex	Description of utilities	47
Index		78

What the heck is hypertext?

Chapter 1

Introduction

First, the analogy!

Let's list everything in your house — garage, basement, closets, boxes, kitchen — to help you find it. That's database indexing.

Better yet, how would you like to walk into any house in town and automatically know where everything was located. That's hypertext indexing.

Intuitive indexing?

This HyperLink manual shows you how to do just that . . . organize complex tangled information into systems of knowledge that are intuitive to all users.

How does hypertext do this?

Critical thinking?

It takes critical reading and thinking to put together information that has little structure. That's hard work . . . causing most people to either skip or ignore your information rather than assemble and comprehend it.

In contrast, if you provide good structure, even with limited additional information, almost anyone can fill in the blanks. Why? Human minds are hardwired for rapid pattern processing, not for critical thinking.

Pattern recognition?

In a nutshell, hypertext puts information in patterns familiar to users . . . so they can easily browse, find, and understand systems of knowledge to whatever complexity they desire. Wow!

Hypertext, the definition of

Communicating the organizing patterns of information

Hypertext is simply adding understandable structures to information so that any reader can rapidly grasp the organizing patterns from any desired viewpoint. With information in this format, users can extract whatever they want from a system of knowledge, regardless of their previous levels of understanding.

Why is structure so important?

Let's begin with today's buzzword . . . taxonomy!

Creating a taxonomy is the difficult part of building hypertext systems. But, once built, such index systems are many times easier to use and more powerful than flat or inverted index systems.

A taxonomy is the classification scheme (or structure of information) for a body of knowledge. Why is a taxonomy or knowledge structure important?

The answer? Knowledge is information that has structure. Think about that.

A taxonomy is not an index or table of contents. Rather, it shows the relationships among each and every idea contained in a pool of information.

What are the principles of hypertext?

Hypertext has three guiding principles: They are:

- index ideas (not just words)
- provide users with minimum-keystroke access to ideas in ways that match their understanding of the subject
- show the relationships of each idea to the larger body of knowledge

Find any information within an average of ten keystrokes without prior knowledge.

What's the value of hypertext?

As the value of information lies in how it is organized, hypertext is a method to extract maximum knowledge from any system of information.

The value of information lies in how it is organized.

You tell me what that's worth . . . if used to organize regulations, specifications, references, requirements, opinions, expertise, or litigation.

What are the system requirements for PC-Hypertext?

Full hypertext on standard computers . . . without MS-Windows or 286 machines.

Oops! I almost forgot the good news. I use a stock 640K 20-meg monochrome PC-XT system to build hypertext systems.

And . . . you can run PC-Hypertext programs on any 256K system using monochrome, LCD, CGA, EGA, VGA, or MCGA monitors. Very interesting!

Chapter 2

In building hypertext systems, you'll use several types of software. The products I constantly use are:

Product	Cost
PC-Hypertext with demo tape	$10
MaxThink 89 outliner/idea processor	$89
HOUDINI 89 system/network processor	$89
HyperLink 89 hypertext utilities	$89
TransText hypertext word processor	$89

All of these hypertext construction products are available from MaxThink. Here's how I use each software package:

Uses of PC-Hypertext software

Your PC-Hypertext disk is important for the following reasons:

Background information

Hypertext overview — PC-Hypertext includes 60 plus essays covering hypertext description, instruction, construction, goals, opportunities — all in a hypertext format. This disk contains your background information about hypertext.

Capabilities

Displays capabilities — To find the best ways to organize and present interrelated information, you need to know all of the features of PC-Hypertext. PC-Hypertext contains many capabilities. Make sure you don't overlook any that might help you or the users of your hypertext systems. This knowledge comes with using the program.

Run time system — As a builder of hypertext systems, you'll have many ways to access the information. However, once assembled, PC-Hypertext is all your customers use and see.

Sample formats

Examples of screen styles and formats — I've used a variety of screen layouts in both my network and ASCII file portions of PC-Hypertext.

While the standard computer screen display and character set is somewhat limited, you can still use eye-movement tools (CAPS, underlines, justification, indents, boxes, spacing) to make your information readily understood by your users. That's style!

Polishing networks.

Working tool — In the later stages of hypertext construction, PC-Hypertext becomes your final ASCII file and ASCII-link editing tool. While in PC-Hypertext, you can automatically call your word processor loaded with the current file. That's useful for polishing your networks.

System validation — In the validation stage, PC-Hypertext is the fastest way to verify that each reference jump is correct.

Easier browsing of large systems of files.

Integration — PC-Hypertext, along with MaxThink,HOUDINI, and TransText, represents and displays structures of relationships. Although structures created by MaxThink or HOUDINI can be loaded into all three programs, I sometimes transfer hierarchies or networks into PC-Hypertext. Its format is often better suited to my needs.

Uses of MaxThink software

Your MaxThink program aids hypertext construction in the following ways:

HyperText node and file management — MaxThink rapidly creates files containing both hierarchical and network (clone) relationships. In addition, MaxThink executes hypertext jumps in ASCII files and between MaxThink files. Finally, MaxThink contains a M2A utility program that automatically converts any MaxThink file (including the clone relationships) into a series of hypertext-linked ASCII files.

I use MaxThink to manage all the hypertext files.

A taxonomy is a consistent method of classification.

Classification — Classification of information is the main process of hypertext construction. MaxThink is very important in the classification of each node, portions of nodes, and groups of nodes from many different viewpoints. The same classification methods also applies to files. The difference between files and nodes?

Not much. My working list of nodes contains a description of the contents of each file followed by the filenames in angle brackets. My working list of files is just the reverse — angle bracket filename <filename> followed by a short description.

Network Editing — I use the TransText hypertext word processor for creating and editing all my ASCII nodes. However, I often use MaxThink for creating and editing the .NET files of PC-Hypertext because the MaxThink editor is more powerful than the editor in HOUDINI. But once edited, I usually transfer the network back to HOUDINI for working on the final appearance.

For hypertext systems of standard operating procedures, MaxThink is my primary editor.

Topic Editing — If my ASCII files contain information using repetitive database formats, I use MaxThink to edit my nodes; then I use the WRITE command of MaxThink to standardize the format (numbering and indentation) of each separate hypertext ASCII file.

Completeness, parallelism, and sequence.

MaxThink philosophy — The MaxThink manual describes the theory of good hierarchies in detail (completeness, parallelism, sequence, and number of subtopics). Hypertext is nothing more than presenting good hierarchical choices from any point in the knowledge system.

Browsing hierarchies is much easier than browsing networks.

Users can't efficiently browse networks because of the dominance of circular relationships. However, users can efficiently browse hierarchies because each topic choice implies subsequent choices. Good hypertext depends on easily understood patterns (hierarchies).

Knowledge is taxonomy mastery.

Second, the MaxThink manual speaks of the relationships between classification, perception, and language. One of the purposes of hypertext is to transfer the language or taxonomy of the knowledge base. Good hypertext also depends on using language in ways that are consistent and meaningful. Why?

Knowledge is information with structure. That's why you can't learn but only verify with relational databases as they do not communicate structure.

Good hypertext can efficiently transfer knowledge. I see knowledge as information (language) with structure. If your language is strongly structured (what it means is obvious), then the structure of your information (hierarchies and links) is easily understood.

Utility programs — Finally, several HyperLink programs use the MaxThink file format for inputs or outputs. For example:

REFALL	Produces a MaxThink format file showing all hypertext jumps of all files in a directory.
IC	INTEGRITY CHECKER program uses MaxThink format as input to validate jump to ASCII files.
PERM	DECISION TREE program uses MaxThink format to create files for building minimum-path decision trees.

Uses of HOUDINI software

MaxThink classifies information by one dimension per level.

Having said all the previous about MaxThink, HOUDINI is still your most important software tool for building hypertext networks. While HOUDINI supports all of the network components such as lists, subsets, supersets, dual paths, rings, and jumps, it is simply too complex for the average brower of networked information. That's better handled by PC-Hypertext.

Houdini classifies information by 5-100 dimensions per level.

With HOUDINI, I index my files (nodes), construct my cross-references, find common node groups, build my networks, and polish the final appearances of my hypertext systems. Thus, the capacity to rapidly build hypertext systems depends greatly on an understanding and mastery of HOUDINI.

Uses of HyperLink software

Short-cut utilities

While HOUDINI remains the central software for hypertext construction, our HyperLink program provides a number of important utilities. For example:

> Programs to create jumps — ANGLE, MP, RMP, MATCH, MARK
>
> Programs to organize information — BIGSORT, WF, PERM, SREF, FS
>
> Programs to analyze jumps — REFALL, IC, REPAIR, INVERT, CONNECT
>
> Programs to maintain hypertext systems — HYPER, ENCODE, TOUCH, DS, CF

Subsequent chapters describe how to use these programs. See the Appendix (pages 47 - 78) for compete details.

Uses of word processors in hypertext construction

As ASCII files are a dominate portion of any PC-Hypertext system, your word processor greatly affects your productivity. Instead of using one of the full featured printing systems, I find theTransText hypertext word processor ($89) ideal for the following reasons:

Faster loads, searchs, and saves

Speed — While in PC-Hypertext, you can call external word processors for instant editing of nodes. If you are not keeping everything memory resident, then you want a word processor that is fast in loading or exiting. Of all the word processors I've seen, TransText holds the speed record in loading itself, loading files, searching for text, saving files, and quitting.

Easy to learn — The last thing I need is another word processor as I already switch between 6 different text-editing tools. However, you can learn TransText in 1/2 hour. In addition you can define the keyboard commands to match those of any other word processor.

Produces ASCII files without format codes

File format — As PC-Hypertext greatly depends on ASCII files, you need a word processor that does not embed special codes into the files. While you can't use fonts and laser drivers, you need commands to rapidly format screen appearances (i.e. indents, columns, right-justification, auto-indents, etc). TransText is strong in these areas.

Dual windows

Split windows — Many times while in your word processor, you'll want to trace ASCII links. Having six hypertext windows open at the same time (TransText) along with rapid file loads is a major aid to the development of hypertext systems you create.

Macros — The obvious macros I've used in my PC-Hypertext are the boxes around important screen information. However, I use macros to enter codes for splitting files and also for building index lists of key words. TransText has very powerful macro commands.

Generally, you can create new ASCII-file hypertext systems hundreds of times faster than new picture hypertext systems unless you have a scanner.

Here's the summary. With ASCII-file hypertext, most information is displayed on the screen using only character graphics (typed text). As such, the techniques to increase readability (placement, accent, flow, interest) are much more limited than with graphic displays. However, even with these restrictions, you can build good-looking ASCII hypertext screens perhaps hundreds of times faster than with CGA or EGA pixel screens.

Simply stated, a hypertext word processor that makes easily understood cross-linked screens differs radically from those designed to shove text to printers. That's why I use TransText.

Using dictionary and thesaurus programs

Stand-alone programs are always more powerful than those that integrate the functions.

Nowadays, it's almost illegal to buy a word processor without the included spelling and thesaurus checkers. But, let's look at these gift-horses in the mouth. Users and reviewers continually report that the included dictionary/thesaurus programs are always a level below the stand-alone software.

For that reason, I like the Reference Set Dictionary/Thesaurus

Reference Set is a memory resident program (45K) that runs in all my most important text tools such as TransText, MaxThink, and HOUDINI. In addition, all three formats of the dictionary program — single word, full screen, and file checking — are compatible with the MaxThink, TranstText, and HOUDINI file formats.

Recommended grammar checker

On Grammar Checkers, I pass my files through GRAMMATIK II for a final check of grammar and style. The program checks for 30 kinds of errors plus suggests changes to improve word usage. While GRAMMATIK II won't make bad writing good, it does make good writing better.

Having covered the tools I use to build hypertext systems, let's consider the PC-Hypertext program itself.

Commands & formats of PC-Hypertext

Chapter 3

Your next step in building hypertext systems is mastery of PC-Hypertext — including operations and information contained in its files. Here's what to look for:

One-page manual

I'm reminded of my MBA classes at Stanford 20 years ago. At that time, they wanted good writing from the students. So . . .

Every class assigned each student a paper that was due every week. The rules were simple. No paper could exceed three pages in length because . . . "if you can't say it in three pages, it's not worth saying."

Learn by doing.

The results? Epic student pain for a couple of weeks; then everyone learned how to quickly write in a short succinct style (good memo training). By the end of the first year, all students could produce reasonably good writing on the fly.

Writing is easy . . . just sit at a typewriter and bleed!

(PS — Stephen King, horror stories best-seller author, says that if you write for an hour a day, in ten years you'll become a very good writer. So, good writing is easy to learn . . . just sit at a typewriter and bleed!)

Build complete knowledge systems that require users to learn only four keys.

To avoid such painful traditions, I've limited the entire user manual to one page. Why? If I can't teach you this program in one page, then it's too complex for a major categories of potential users.

For that reason, I've kept PC-Hypertext simple enough to learn in two minutes as my hypertext systems are often the first computer experience for many users. (See the next page.)

The ways you access knowledge identifies your understanding of the information contained.

However, since you're a hypertext developer, you'll need more than this one-page summary. PC-Hypertext includes many sophisticated capabilities that are only apparent if you know all of the commands. With that, let's review the commands of PC-Hypertext.

Description of a PC-Hypertext topic screen

*Cursor — **Up/Down arrow** keys move the cursor up or down the list. Press **Right-arrow** key to select the cursor topic.*

Description of the knowledge system

Last topic selected

*Press **Left arrow** to display higher levels of topics*

Topic list (numbered)

Listed commands
Spacebar moves highlight
***Enter** selects highlight command*

*PC-Hypertext contains seven different help screens. At any point, press **F1 key** to display the pertinent help screen.*

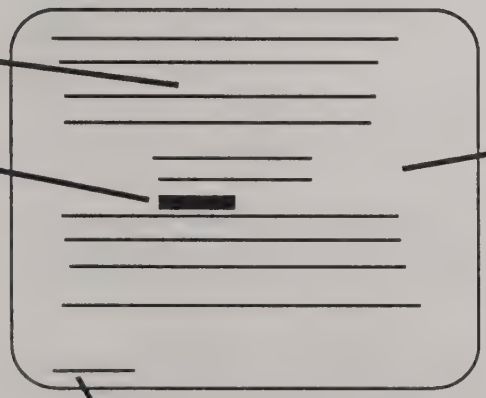
File information

*Display status — **F2 key** expands/collapses text*

*Press **Right arrow** on topics containing <filenames> to display that ASCII file (shown below)*

*Press **F5** on either display to mark topics for later use. MARKED command includes options for printing or saving selected marked topics or files.*

Description of a PC-Hypertext ASCII-file screen



ASCII file text

*At any point, press the **Left arrow** to return to the above network display*

Highlight <filename>

*Use **Up/Down arrow** keys to display other portions of this file*

***Up/Down arrows** highlight <file> in text*

*Press **Left arrow** key to return to the previously displayed ASCII files*

***Press Right arrow** key to shift display to highlighted <file>*

Filename / status

*PC-Hypertext contains seven different help screens. At any point, press the **F1 key** to display the pertinent help screen.*

Help screens list all the commands

All the commands of PC-Hypertext are listed in the HELP.HYP file. PC-Hypertext includes seven difference help screens. Here's how to see all of these screens.

Remember to press ESC ESC after each screen to return to the MAIN menu:

Seven different help screens

Action	Results
Press: F1	Displays MAIN menu helps
Press: O F1	Displays OPTION menu helps
Press: M F1	Displays MARKER menu helps
Press: G F1	Displays GOTO menu helps
Press: Alt-K P F1	Displays Line editor helps
Press F1 on an ASCII file	Displays ASCII-file helps
Press F1 on a graphic file	Displays graphic-file helps

Types of hypertext jumps

Another approach to understanding PC-Hypertext is to look at the types of hypertext jumps the program supports. For example:

The arrow keys are like exploring caves and automatically unwinding twine to keep from getting lost.

Smooth network — While the initial screens of PC-Hypertext appear as hierarchies, the links between topics are a network of back jumps, loops, dual paths, and super-sets. It is the hierarchical display that makes the network appear as a smooth surface easily traversed by arrow keys alone.

Networks provide transitive links to rapidly move to information.

Network ASCII jumps — The hypertext jumps between a network file and ASCII files are also transparent to users. Again, the Right/Left arrow keys smoothly switch between the two kinds of files.

ASCII files provide explanatory links to expand or clarify information.

ASCII file jumps — The more obvious hypertext jumps in PC-Hypertext are ASCII files containing <filenames>. These kinds of jumps use a different selection process (Up/Down, Right/Left arrows).

Complex file jumps — PC-Hypertext also includes capabilities to jump to the top or interior points of networks.

PC-Hypertext permits graphic browsing without using a mouse.

Graphic file jumps — PC-Hypertext supports hypertext jumps to display PC-Paintbrush or ColorTX files.

Use PC-Hypertext as a universal shell to access programs, files, and, knowledge.

System hypertext jumps — In calling DOS or other programs, PC-Hypertext saves its status, writes a batch file containing the system call and exits from memory to run the batch file. Once completed, the program reloads itself to display the previous information.

You can use these system hypertext jumps to run DOS commands or multiple programs from within PC-Hypertext. At the outer limits, you can have the called programs modify ASCII or network files of PC-Hypertext to create a dynamic, self-modifying hypertext structure. Complex, but it works!

Information contained on the PC-Hypertext disk

The PC-Hypertext Demo disk contains information that's important in the design and construction of your hypertext systems. For example:

Contents

ASCII-file data — The files contained in PC-Hypertext provide useful hypertext information such as goals, philosophy, design, coming features, etc. I'd recommend that you print these files, then critically read each one. Critical reading means using a yellow marker to highlight points that are important or new to you.

Formats

ASCII file formats — Scan the files on the disk to review the different ASCII formats used — solid text, left and/or right indented text, right-justified text, boxed text, margin comments, numbered lists, and tables. One of the arts of hypertext is creating an information style that appears ordered and interesting on a monochrome or CGA display.

In addition, my demo Hypertext system consists of fairly linear information (typically multi-screen files). However, in other hypertext systems, I've successfully used smaller ASCII files (single screen maximum file size) and standardized my displays.

Networks

Network formats — While the network portion of my PC-Hypertext demo is strongly hierarchical, I've also created other hypertext systems to access information from many different networked dimensions. Be aware that if your information is multi-dimensional, the network portions of your systems will be very different from the formats I used with my PC-Hypertext demo disk.

Philosophy

Hypertext theory — Finally, various ASCII files of my demo system describe the theory and philosophy of hypertext. Make sure you understand the reasons for (and against) hypertext, databases, and expert systems so that you use the appropriate technology for information problems you wish to solve. While hypertext is powerful, it is not intended to be the universal solution.

Special features

PC-Hypertext also includes several unusual features. They are:

How to organize your files

Auto sub-directory — While MS-DOS allows 64,000 files per directory, file access speed becomes unworkable above 200 files per directory. With any file access, PC-Hypertext first looks for the requested file in a directory consisting of the first three letters of the filename. If that directory does not exist, then the program searches the current directory for the requested file. This approach lets developers create either flat or hierarchical file systems.

Having built a number of systems, I make the following recommendations:

Recommendation

> If your system will ever contain more than one hundred ASCII files, name your files sequentially (AFILE01, AFILE02, . . . , AFILE99); then start a second set (BFILE01, BFILE02, . . ., BFILE99).
>
> Create sub-directories AFI, BFI, etc. to contain your files. However, in PC-Hypertext, use only the filename, not the full path for your hypertext jumps.
>
> When you ship hypertext systems, remember that DOS only allows 112 files per disk, whether 360K (5-1/4) or 720K (3-1/2). For that reason, I ship my larger systems with one disk per directory.

Options menu

Options menu — The word processor call in the OPTIONS menu includes these options: (My PATH command includes TransText.)

Call your word processor from within TransText

> "#" means load the current file name
>
> "@" means load the current filename using the first three letters of the file as the path. For example:
>
> | | |
> |---|---|
> | TransText # | Loads the current file from HYPER directory |
> | TransText @ | Loads the current file from a subdirectory |

Learning tapes

Audio/visual feature — The ALT-K command, which builds or plays back a log of keystrokes, includes an unusual feature. The program records both the keystrokes and the delays between each keystroke.

Make cassette tapes to aid learning

Here's the reason why. I can make audio tapes (cassette format) containing instructions for use of PC-Hypertext such that during the playback, both the audio tape and visual actions on the computer screen are in sync. Neat!

Normally, I wouldn't want to mention any problems with PC-Hypertext. However, here's one of my major concerns with this program.

Problem of rapid evolution

PC-Hypertext is evolving very rapidly. I'm constantly adding features and improving the user interface. One of the files I included with PC-Hypertext mentions many of my planned changes to the program. But, it doesn't say when these changes will occur (they actually happen weekly).

Solution to rapid evolution

For that reason, call me as you get ready to release your own hypertext systems to obtain the most recent version of PC-Hypertext to ship to your customers. I'll include a list of changes so you'll always know the current capabilities of the program.

What are the changes that are coming? See Chapter 13 (page 51).

Necessary files to run PC-hypertext

Chapter 4

Now, let's cover the details of PC-Hypertext itself.

A running PC-Hypertext system requires the following ten types of files to be present. Here's what each does:

HYPER.EXE	is the main program of PC-Hypertext. This is the file that I'm constantly improving and expanding. For that reason, make sure you have my latest version of the program to include with your knowledge bases.
TITLE.HYP	contains the title screen to PC-Hypertext. This is the first file PC-Hypertext looks for during the loading process. Thhis file can be ASCII or Color-Tx in format.
MESSAGES.HYP	contains the program prompts and error messages. This is the second file PC-Hypertext looks for during the loading process.
HELP.HYP	contains the help screens to PC-Hypertext. This is the third file PC-Hypertext looks for during the loading process.
CONFIG.HYP	contains the setting to the options screen and is the fourth file loaded by PC-Hypertext.
FILELIST.HYP	contains an encrypted list of the networks files and the title line (screen line 1) of the MAIN menu. Our ENCODE program (described on page 18) reads an ASCII file containing this information to help create your FILELIST.HYP file.
.NET files	You need to include at least one network file in order to run PC-Hypertext. For example, the network file in my demo release was named HYPER.NET. Create your network files either using MaxThink or HOUDINI, then run our M2H or H2H programs (see page18) respectively to convert these files into the PC-Hypertext .NET format.

ASCII files	While PC-Hypertext will run without any ASCII files, each ASCII file functions as a hypertext node in PC-Hypertext. As a consequence, you may have hundreds or thousands of ASCII files in the hypertext systems you create.
CTRL.COM	Batch file return call. If you use PC-Hypertext to run DOS commands and other programs, this file is necessary to return control to PC-Hypertext at the end of the batch file process. For that reason, don't leave it out.
REF	The REF command on the MAIN menu either displays the ASCII file named REF or runs a program named REF. The REF command provides convenient ways to access your information from different viewpoints.

This summarizes necessary PC-Hypertext files and the reasons for their inclusion on knowledge bases you create.

Files for building your own system

To get the necessary files out of PC-Hypertext (in drive A) to build your own system (for example in drive B), use these commands:

Files from PC-Hypertext

COPY A:*.EXE B:	Gets the HYPER.EXE file
COPY A:*.HYP B:	Gets the TITLE, HELP, MESSAGES, and CONFIG files
COPY A:*.COM	Gets the CTRL.COM file

The files you'll create for a working hypertext systems are:

Files you create

File	**Purpose**
FILELIST.HYP file	Identifies the legal network file(s)
.NET file	Network that leads to ASCII files
ASCII files	Serve as hypertext nodes
REF file	Alternate way to acess information
Color-Tx files	Displays textual graphics/colors
PC-Paintbrush files (PCX)	Shows CGA, EGA, VGA, formats

Now, let's consider your hypertext working environment.

Setting up shop

Chapter 5

This chapter describes my hypertext working environment and the batch files used in the construction of PC-Hypertext systems.

Recommendation

For example, I use the following file directories:

HOU directory	Contains my HOUDINI program
MAX directory	Contains my MaxThink program
TR directory	Contains my TransText word processor
HU directory	Contains all my HYPERLINK utility programs. In addition, I keep in this directory my MaxThink or HOUDINI data files used to create my PC-Hypertext network files (source networks) as well as the .NET files (object networks).
FILES directory	Contains my ASCII files. I modify the ASCII portions of my network using files in this directory, then I copy the modified files to my HYPER directory. This keeps the .BAK files from my word processor out of my working directoryand lets me gradually merge my completed ASCII files into my working hypertext system.
HYPER directory	Contains your hypertext system consisting of your network and ASCII files along with the selected .EXE, .HYP, and .COM files from PC-Hypertext.

How to set up your directories

The previous page descirbed my different directores. Use the MD (make directories) and COPY (copies files) commands of DOS to create your directories, then copy the files from my distribution disks.

From the PC-Hypertext demo disk, copy the key files (HYPER.EXE, .HYP files, and CTRL.COM) to your working (where you'll begin creating your hypertext system).

Contents of the HyperLink disk

Disk two of the HyperLink package contains contains all the utility programs. These program are in an .ARC format which compresses them into a single file. To recover the utility programs from the HyperLink Utility disk in drive A:, do the following at the A: pompt:

Action	**Results**
Type: ARCE A:UTIL.ARC C:\HU	Utility programs are extracted to directory C:\HU

The resulting files created in the C:\HU directory are:

Making hypertext nodes	FS	Makes ASCII files into hypertext nodes
	M2A	Converts MaxThink files to ASCII file hypertext
	H2A	Converts HOUDINI files to ASCII file hypertext
Making hypertext jumps	ANGLE	Makes hypertext links to all files in a directory
	MATCH	Makes hypertext links to defined phrases
	MARK	Makes hypertext jumps to files by key ideas
	MP	Makes hypertext links to words in a directory
	RMP	Makes hypertext links to non-garbage words
Organizing information	WF	Determines word frequency of words in a directory
	BIGSORT	Organizes hypertext links into common categories
	SREF	Collects ideas into hypertext categories
Link analysis	IC	Checks integrity of all links
	REFALL	Shows all hypertext jumps from a file
	REPAIR	Automatically correct error in links
	CONNECT	Shows generations of links with specified files
	INVERT	Shows all hypertext jumps to a file
System maintaince	CF	Identifies unwanted changes in a hypertext system
	DS	Records status of all files in a hypertext system
	TOUCH	Sets all file date stamps to a specified value
Special programs	ENCODE	Creates security file for running PC-Hypertext
	TRK	Converts keystroke log files to hypertext format
	M2H	Converts MaxThink files to a .NET network file
	H2H	Converts HOUDINI files to a .NET network file
	PERM	Builds expert systems/decision trees using MaxThink
	AN	Inserts user name into FILELIST.HYP file

Chapter 6

The network portion of PC-Hypertext normally uses a rooted file. "Rooted" means that from the top node in the network, there is a path to every other node in the network. For example:

All MaxThink files are rooted as the program only allows for one TITLE topic.

A "rooted network" has only one topic with no input links

HOUDINI files, however, may or may not be rooted. While you can use non-rooted files in PC-Hypertext, you need to include network jumps in the network to reach some of the topics. For that reason, I recommend that before translating a HOUDINI file into the PC-Hypertext format, first check to see if it is rooted.

Look at the MAIN. If a topic has no input links, then you know the HOUDINI file is rooted with that topic at the top of the network.

Making your network (.NET) file

As mentioned previously, I keep my HOUDINI and MaxThink files that form the network portion of the PC-Hypertext system in my HYPERU directory. To make a running PC-Hypertext system, first do the following:

Action	**Results**
Create a rooted HOUDINI file named TEST.HOU	Builds network for PC-Hypertext

Make the first HOU topic rooted, then save your rooted file to the HYPERU directory. I use a .HOU file specification for my network file names. For this example, use the filename TEST.HOU for your HOUDINI file that reflects your hypertext network.

Differences in Houdini and PC-Hypertext file formats

What's the difference between a HOUDINI file and a network file? The link information in HOUDINI files is in ASCII format. The link information in PC-HYPERTEXT files is in binary format.

Benefits of the binary file format

Why the binary format? Two reasons — binary files load twice as fast and binary coding of links provide some protection against "easy theft" of your coding of hypertext relationships.

Action	Results
Type: CD \HYPERU <enter>	Change to the HYPERU directory
Type: H2H TEST.HOU <enter>	Runs the batch translation program which converts the HOUDINI file TEST.HYP into the PC-Hypertext network file TEST.NET.
Type: Copy *.NET \HYPER <enter>	Copies the network files to the HYPER directory

Or use M2H with a MaxThink file

Every time I change my network file in HOUDINI, I run this sequence of commands.

Making your FILELIST.HYP file

PC-Hypertext won't load any network file unless it is listed in the FILELIST.HYP file. To make this file, do the following:

Action	Results
Type: CD \HYPERU <enter>	Changes to the HYPERU directory
Type: TYPE FILES <enter>	Lists the FILES file

H2H and M2H create network files

The first line of this ASCII file appears at the top of the MAIN menu screen of PC-Hypertext. Each subsequent line lists the converted HOUDINI or MaxThink files into your system.

ENCODE reads FILES to create FILELIST.HYP

Use your word processor to modify my FILES file to include your own title line and your network file TEST.NET. Make sure your word processor includes carriage returns and line feeds at the end of each line so as not to confuse my ENCODE translator program.

Do the following to convert FILE and then run your system:

Action	Results
Type: ENCODE FILES <enter>	Read FILES file to create the FILELIST.HYP file
Type: COPY FILELIST.HYP \HYPER <enter>	Copies the resulting file to the working hypertext system directory
Type: CD \HYPER <enter>	Change to the HYPER directory
Type: HYPER <enter>	Runs PC-Hypertext with your new network

Using batch files to automate the process

I use a batch file to automatically run my entire network modification process. When I exit HOUDINI (having first saved my data in my HYPERU directory under the name TEST.HOU), I type NEW. The NEW.BAT file contains:

My NEW.BAT file

```
CD \HYPERU
ERASE TEST.NET
H2H TEST.HOU
COPY TEST.NET \HYPER
CD \HYPER
HYPER
```

With my changes loaded, this batch file puts me directly in PC-Hypertext .

Using MaxThink networks

The same process used to convert HOUDINI files into PC-Hypertext networks also applies to MaxThink files. If you use MaxThink files, then use the translator program M2H instead of H2H.

H2H = Houdini to Hyper
M2H = MaxThink to Hyper

When you translate MaxThink files directly into PC-Hypertext, the link structure works as intended, but the appearance may surprise you. For that reason, I often create my structure in MaxThink, load my MaxThink file into HOUDINI to polish the appearance, then transfer the HOUDINI file to PC-Hypertext.

Using multiple network files

Both the H2H and M2H translator programs will do wild-card file conversions (i.e., TEST*.HYP converts to TEST1.NET, TEST2.Net, etc.).

If you are using multiple networks in a PC-Hypertext system, change your batch file to erase all *.net files, convert all *.HOU files, then copy all *.net files to your working directory.

Don't confuse me!

Why do you erase the *.net files? Because both the H2H and M2H programs abort rather than overwrite a file with the same .NET name.

Adding word processor links

Chapter 7

Once you have a network up and running, you're probably ready to start adding ASCII files along with links between each.

How to create spaghetti hypertext

That's fun as long as you're working with less than 20 files. Using your word processor, you can easily create files, add <filename> jumps, and cross-link it like mounds of spaghetti that exceeds your capacity to follow or organize.

After creating several such messes, you've come to the conclusion that there must be better ways to organize the hypertext links between ASCII files. You're right!

Here are some hypertext lessons that only experience teaches:

1) Don't build ASCII files on the fly to satisfy a needed link

2) Don't cross-link ASCII files on the fly to build a network

Hypertext construction: Take care of the pennies and the dollars take care of themselves

Why these rules? Because the systems of cross-links that you build will always exceed your ability to comprehend the relationships in their entirety. So, let's consider an alternate approach.

I use a very systematic process to build my hypertext systems that is exactly backwards from what I've taught you so far. How come?

Hypertext construction: It's like a multi-axis BINSORT

Well, the last thing I do is build my network .NET file. The only reason I showed you how to use H2H and ENCODE is to let you experiment with your own hypertext network before getting serious about building workable knowledge systems.

This is a short chapter!

Few people can mentally organize large amounts of information from multiple dimensions without a systematic process

To get a clear understanding of how not to do hypertext, try closing this book and build a small hypertext system using a simple HOUDINI network that links to a dozen or more ASCII files.

After a couple of hours of effort, you'll continue this book with a new appreciation for the efforts needed to build good hypertext!

Collecting your knowledge

Chapter 8

Either you create the knowledge contained in a hypertext system or it is collected from others. Let's consider each.

Conversion hypertext

Collected knowledge — Fortunately, most hypertext construction centers on structuring and indexing information that already exists. Your contribution is creating organized access.

Original hypertext

Created knowledge — I created the files that I used for my PC-Hypertext demonstration disk. While creating the knowledge for your hypertext system is not typical, if the information doesn't exist except in your mind, then it's your job to write before you consider hypertext. That alone stops many hypertext systems before they're ever started.

Easy hypertext

Easy hypertext systems use knowledge that is already organized in printed and indexed formats. Examples of easy-to-construct hypertext disks are systems of indexed information found in books, articles, laws, regulations, or specifications. In these formats, hypertext construction consists of:

Cheap and dirty methods of hypertext

1) Reversing the index. Sort the existing index by page number

2) Making nodes by index reference. Split the text into nodes (ASCII files) by each index reference.

3) Cross-linking the nodes. Put in the <filename> references to other passages mentioned in each index entry.

If the original index contains single references for each topic, the resulting system tends to be more hierarchical than hypertext. However, if the original index contains multiple references for each topic, then the resulting system becomes reasonably good hypertext.

No-brain hypertext builds indexes instead of taxonomies.

Comment: I'd call this no-brain hypertext. That is because your major contribution has been embedding the index choices in the text as opposed to leaving it at the end of the book. Nevertheless, using PC-Hypertext to instantly jump to all related topics instead of thumbing through an index offers significant advantages that justify the hypertext effort.

Average hypertext

More often than not, even though your information exists in printed form, the quality of the indexing tends to be poor. In these cases, you've got work to do.

Put the index into HOUDINI as a starting point for building a taxonomy.

For example, you need to find, then understand the taxonomy (language classification schemes) of the knowledge area; then you can use this language to index the information.

This abstracting process extends several mental levels beyond merely the re-indexing of information. Why? Because abstracting simply requires critical thinking skills, which also explains why hypertext is difficult to construct.

Complex hypertext

Occasionally, you'll find hypertext opportunities where the information exists in printed forms, but where no index exists. This is where you earn your keep as the task becomes the creation of the taxonomy (classification schemes or access methods) for the subject. This takes conceptual thinking.

How often does this occur?

While the word "taxonomy" normally applies to biological systems, it is central to systematizing any field of knowledge.

More often than you might imagine — for example, it occurs in policy manuals, operating procedures, sales and marketing techniques, and consulting approaches. In fields that change rapidly, the magazines may be one year and the books from three to five years behind current thinking. With changing information, indexes or taxonomies often don't exist.

Formats of knowledge

However, besides the quality of existing indexes affecting the complexity of building hypertext systems, the format of the collected knowledge also can create grief for developers. For example:

ASCII files

Disk format — The most desirable beginning format for adding information to hypertext is ASCII files on disk.

Easy scan text

Typewritten text — If the knowledge comes from typewritten documents, most desk-top publishing centers have low-cost scanners that easily convert typewritten text into ASCII files. In my area, such services costs $6-10 per hour with scanning rates of perhaps 20-40 pages per hour.

Difficult scan text

Typeset text — If the knowledge comes from a book, newspaper, or magazine, a more sophisticated scanner is needed to convert the information into an ASCII file. Again, in my area I've seen rates of from $2 to $5 per page, with productivity of from 5 to 20 pages per hour.

As a short side comment, we have two OCR machines that rapidly convert both typeset and typewritten information into ASCII files. I'll let you use these machines for almost zero cost. Why? I'd like to help PC-Hypertext developers get started in building knowledge systems. Besides, I am interested in peeking at the kinds of information organizing opportunities others see.

Scan it to a graphic format

Graphic knowledge — If the knowledge comes in a graphic format, you can either describe it textually, convert it to a device-independent format, or wait for our next major release of PC-Hypertext, which includes full graphic conversion and browsing.

Hypertext and expert system construction share the same problems of identifying knowledge.

Handwritten or verbal knowledge — If the knowledge is in handwritten or verbal formats, I don't even want to talk about it. You've just got a lot of work ahead of you.

Copyrights

One of the serious problems with hypertext is the issue of ownership of material. Generally, most everything printed or published is copyrighted, with the original author automatically retaining the rights to derivative work. I'll let the courts decide if hypertext indexing is or is not a derivative work.

What is a derivative work?

However, everything the state or federal governments publish — laws, regulations, specifications, rulings, filings, information, directories, books, pamphlets, articles — is in the public domain. This information is available to anyone who wants to organize it into hypertext knowledge systems. Considering the government has 15,000 books alone in print, hypertext opportunities are not particularly limited.

Govermental publications are a hypertext gold mine.

If that's not enough, the unorganized information within a business or profession (i.e., testimony, articles, activities, directories) awaiting hypertext staggers the imagination.

Court battles are won or lost in the trenches of information mastery.

Late notes

My WALL STREET JOURNAL (2/25/88, page 23, col. 2) carried a story saying that the manuals for future weapon systems would be computerized. If you can put up with the bidding paperwork, that's a way to build your understanding of MaxThink, HOUDINI, and HyperLink into your own defense-contracting firm. Why do it?

25% of the weapon manuals are always out of date — 6% of the military budget is in documentation.

Just as major firms now exist in conventional publishing (software, print, film, records, etc.), the future major firms in CD-ROM and hypertext publishing are now people like yourself who have vision and skills to structure knowledge. Besides, government-contract work earns good money while you polish your hypertext skills!

If you include training costs, poor documentation clearly becomes the most expensive lifetime component of any hardware system.

Creating ASCII hypertext nodes

Chapter 9

The longest intellectual journey still takes a step at a time.

In constructing large hypertext systems of thousands of nodes and links, the first step is to break the collected knowledge into numerous ASCII files.

This step is called "decomposition," which is the taking apart or disassembly of the information. In practice, this means splitting large ASCII files into small files, one for each idea.

To split ASCII files, let's look at our FS (File Splitter) program.

Action	**Results**
Type: CD \HU <enter>	Changes to the HU directory
Type: FS <enter>	Runs the FS program

Surprise! The program displays the instructions for using it. In fact, all the HyperLink utilities display their instructions without filename parameters.

Take a break to see what's on the HyperLink Utility disk.

As I imagine you're curious about the files that make up HyperLink, type the name of each program to see what happens. In later chapters, I'll cover each in detail. But for now, do the following:

Action	**Results**
Type: H2H <enter>	Displays instructions to the HOUDINI to PC-Hypertext file conversion program
Type: M2M <enter>	Displays instructions to the MaxThink to PC-Hypertext file conversion program
Type: ENCODE <enter>	Displays instructions to the program that creates the FILELIST.HYP file.

	Action	Results
See Chapter 10	Type: MARK <enter>	Loads the program for marking key words in files
	Press: F1 key	Displays help files to the MARK program
	Press: ESC F8 keys	Leaves MARK program
See Chapter 10	Type: SREF <enter>	Displays instructions to the program that builds the sorted-phrase index to all the files.
See Chapter 11	Type: REFALL <enter>	Displays instructions to the program that locates all the hypertext jumps in all files
See Chapter 11	Type: IC <enter>	Displays instructions to the program that verifies that all the hypertext jumps reference files that exist
See Chapter 12	Type: PERM <enter>	Displays instructions to the program that builds decision trees paths (i.e., resolves path permutations)
See Chapter 13	Type: DS <enter>	Displays instructions to the program that records the status of all files in a knowledge system
See Chapter 13	Type: CF <enter>	Displays instructions to the program that identifies changed files in a knowledge system
	Type: TOUCH <enter>	Displays instructions to the program that changes date/time information in files

Having taken a peek into the hypertext toolbox, I'll make one obscure comment if you can't wait for my instructions — run the above programs (along with the other included utilities) in the same directory as the referenced files. Why? Because PC-Hypertext can include automatic subdirectories which these utilities also check for and process if necessary.

Let's get back to the FS (file splitter program).

What does FS do?

The FS program splits large ASCII files into numerous smaller ASCII files that will become hypertext nodes.

How does FS work?
The FS program needs two filename parameters -- the input filename (your large ASCII file) and an output filename (a summary list containing the filename and description of each created node).

How do you use FS?
First, load the large ASCII file into your word processor. Then add a new line of text in the file to indicate the start of each node and text to skip (if necessary). Once you've marked all the locations to split the file, save the file; then run the FS program.

What codes does the FS program recognize?

==== <filename> description	Starts a new file
==== description <filename>	Starts a new file
==== skip	Identifies text to skip
==== write	Identifies text to append

Which is the best code format for starting new files?

I use the description <filename> format because I'll later load this information into Houdini.

I prefer to put the node description first, followed by the filename. I use this sequence (description then <filename>) in each of my network nodes that lead to ASCII files. If I need the opposite format (perhaps for a REF file), I build a macro to create a file with <filename> followed by the description.

How long is the description?
The FS program looks for a carriage return. While you could set the right margin at 255 characters, I limit my descriptions to perhaps 40 characters. However, I set my tabs for the <filename> entry so that they all align on my output list.

What are the best descriptions?
I like to make my description3 in the form of solution statements (i.e., How to write good descriptions, How to enter codes for splitting files, etc.).

Where do you enter these codes?

How to read critically

I critically read the text by mentally summarizing the contents of each paragraph. When my summary description shifts significantly, I enter the codes that create a new file. Each file should define, contain, or explain only one idea unit important to the larger system of knowledge.

How high is up?

What are the mistakes one might make in using FS?
If your hypertext node is too broad (too large or on a multiple screen), then the links to it do not provide specific answers. If your hypertext node is too small (single sentence or paragraph), then links to it do not show the context of the information contained.

How long does it take to run FS?
In knowledge areas that you know very well, you can create hypertext nodes (split files) at the rate of perhaps 100 per hour.

What are the alternate ways to split files?
You could use the block write command in your word processor to save each marked block to a file. However, this process would not create the FS output list of each of the created filenames and their description.

What is the output list use for?
The output list (filenames and description) is used in the MaxThink, HOUDINI, or PERM programs to create the network paths that lead to each file. In addition, this list is used to manage the inclusion of the ASCII hypertext jumps in each file.

What filenames are best to use?

That's the most important question asked so far. Here's why.

While PC-DOS allows up to 64,000 files per directory, even the fastest 386 machines run like molasses with more than 200 files in a directory. So . . .

If your hypertext system is less than 200 files, name them anything you want (knowing that the floppy disk only allows 112 files per directory).

However, if your hypertext system will ever contain more that 200 files, then you need to use sub-directories. Surprise.

PC-Hypertext automatically looks into a sub-directory consisting of the first three characters of any filename. If the sub-directory does not exist, then the program searches the current directory for the file. Given all this information, I recommend the following filename conventions:

1) Make all your filenames the same length (i.e., file01, file99)
2) Limit the number of files in each sub-directory to 99 files
3) Sequentially number the files in each sub-directory
4) Create a separate directory for each major area of knowledge (i.e., Afile##, Bfile##, Cfile##, etc.)

Why not use meaningful names for filenames?

Using meaningful names really means that the names are important to you. The users of your system will seldom reference information by the filename. However, if someone wished to duplicate your work, a meaningful filename system totally un-locks it. Ooops, the big No-no!

That raises the question of piracy of your hypertext system for use, or piracy of your hypertext system to create a competitive knowledge system. There's another reason to use numbered files instead of names. Ugh!

Why I use numbered filenames

Finally, in lists of 300 to1000 files, an eight-character filename still doesn't have much meaning. The output list from FS (containing each filename and description) provides a much better file reference system.

What are the implications of the file splitter program?
The FS program implies that you'll build hypertext systems in a batch process. A batch process means that you complete each task whether splitting files, building access references, text proofing, or linking before starting the next one. The FS program supports just one of these tasks.

However, if you want to combine all of these operations (to perhaps reduce the number of times you look at the text), you can build macros in your word processor that imitate the processes of our FS and MARK (next chapter) program. My advice? Save that for your next system even if your word processor has that capability.

What word processor should you use?
I use TransText for most of my hypertext work. If you use another word processor in the splitting process, use one that creates an ASCII file (without adding any codes to the text) and one that puts a carriage return (ASCII 13) at the end of a line.

Why is FS limited to splitting 128 files at a time?
The program checks each filename you code for creation to see that it is unique. If it duplicates another filename, the program identifies the error, then quits. Solution: Correct the filename using your work processor, then run FS again.

Anything else?

Add enough information so that each hypertext file retains meaning in an isolated context before adding links to other files.

Yes. The purpose of splitting text is to create isolated pools of explanatory information that make sense by themselves. However, since the original linear text seldom is organized that way, you'll need to add information to each split file.

For a minimum, each split file should include:

A <filename> reference to the proceeding file
A <filename> reference to the subsequent file
A source reference (title, section, page number)

In addition, you may wish to include a short introduction to the specific information contained. Surprise! Hypertext construction includes a lot more word processor work than most people can imagine.

Now, it's your turn

The next chapter needs the following files -- file1, file2, file3.

Right now, use the FS program on any ASCII file to create these files.

Cross-linking ASCII files

Chapter 10

Cross-linking information is perhaps the hardest part of hypertext construction. It requires that you identify and find all the important pieces of your knowledge area, then construct links from each piece to every other related piece.

Specifically, each file you created probably contains several topics with the file that logically references information in other files. How do you build these links? That's the subject of this chapter. Let's start.

What kind of links?

In general, hypertext contains two classes of links — explanation links and choice links. Here's the difference:

Single hypertext jumps generally link to ASCII files

Explanation links — If you find a cross-reference jump <filename> in a paragraph, this is typically an explanation link. Why? Because it should provide expanded information on the word(s) or idea(s) associated with the reference. Generally, users explore all explanation links because there's little hint of what lies behind the jump.

Groups of hypertext jumps generally are best handled in network files

Choice links — If you find a list of cross-reference jumps <filename>, these links offer users a choice of hypertext jumps. Typically, users select the best choice instead of exploring all choices. If the list of jumps is well organized, the users can predict where each jump leads to. Here are the differences:

"Transitive links" rapidly move through information

The links in ASCII files tend to offer explanation, expansion, or commonality while the links in network files tend toward selection, resolution, or taxonomy (classification schemes of the knowledge area). If the word "taxonomy" didn't choke you, here's a bigger word — transitive. My network links are transitive; my ASCII links aren't. Wow!

Chapter 12 covers network linking while this chapter covers explanation linking. Now the details.

What is the problem?

To make explanation or commonality links, you'll need to locate all the ideas in all the nodes, then identify all information that shares some relationship with each idea. Specifically, you need a list of the ideas along with a list of files containing similar ideas. Ugh!

What do you do with this list?
Assuming I have this list, I look at each file containing a particular idea, then see if any portion of that file should contain a cross-reference to another file that provides further information.

How do you physically make these links?
I print out each of my ASCII files. Using my list of files associated with each idea, I spread out on a table only the printed files that contain one particular idea.

I carefully read each file to see how it relates to each of the others. If any of the comments in a particular file are expanded on in another file, I pencil in the <filename> reference.

How many files on each ideas?

How many cross-references do you make?
Typically, I like to find three to seven files on each idea. If I find more than seven files, the ideas may be too broad; if I find only few files containing the same ideas, the concept may not be important.

"Star" linking prevents circular hypertext

What type of cross-references do you make?
I like to make "star" references. I locate the most important file out of several on a particular subject. I include a single cross-reference to this file in each of the remaining files. In addition, I include in the lead file references in the text or footnotes to each of the remaining files.

Why are star cross-references important?

With seven files on a single topic, you could make 42 cross-references (each of the seven refer to the other six). Besides being both time-consuming and circular, such linking provides all paths to the information, but not necessarily the best path.

Do you make 42 or 12 links?

Using star links with seven files, you make only 12 cross-links (center links to six other files and each of the six link to the center file). Yet, from each file, you can reach any of the others in the set within two hypertext jumps.

Why do you build your ASCII cross-links on paper?
In the link construction process, I'm trying to integrate a common-idea set of nodes using a minimum set of links. To do this, I need to see all of the nodes simultaneously. I can't do this on a computer screen.

Why I make 18 instead of 12 or 42 links.

Using the previous example of seven files on the same idea, I actually read each file, looking for possible links between the remaining six. I will consider each of the 42 possible linking relationships, but make perhaps only 18 links (the minimum set of 12, plus one extra link so each file contains two further branches).

Preventing link "overload."

However, there's another reason for penciling in my links on the printed copy of each file. I've just described the process for linking hypertext nodes on a single idea. Yet, each node can contain multiple ideas. As I build my links one idea at a time, I often find pivotal nodes that could contain hypertext jumps in every sentence or paragraph. In such cases, I choose the dominant link and remove others (unless it breaks a star relationship).

Making systems that are easy for others generally means hard work for you.

Complex? Yes, because you balance constraints of the minimum and maximum number of links to a node against conflicting opportunities for jumps. I find my best approach is to pencil in all the possible jumps, add my constraints to balance the system, then enter all my <filename> jumps at one time into each file.

How do you build your master list of possible links?

I thought you'd never ask.

I build my index of ideas contained in each topic using one of three different methods — the MARK program, HOUDINI, or my word processor. Let's cover each in turn.

MARK program

The MARK program was my first approach to identifying and indexing all the ideas across a large number of files.

MARK is likes a word processor with only block marking commands.

The program functions like a word processor. It loads files, lets you rapidly mark key words or phrases in each each file, add synonyms, summarize the context of the total file, and save this information for inclusion in a master index of ideas. Let's try it:

Action	**Results**
Type: CD \HU <enter>	Changes to the HU directory
Type: MARK <enter>	Loads the MARK program
Type: F7 file1 <enter>	Loads file1

Ideal explanatory files have many input links and isolated output links.

Ideal transitive files have many input links and 5-10 parallel output links.

Your task is to mark perhaps four key words or phrases in this file. Use the arrow keys to move the cursor to the beginning of one word, press F5 to turn on the marker, press Ctrl-End to extend the highlight to the end of the word, then press Enter to add the marked text to the index list.

Now the problems!

Vocabulary control?

One of the problems with this approach to indexing ideas by the words contained in the text is vocabulary control. While the same idea may be described in several different ways, the sorted list of words and phrases does not reflect this commonality.

A second problem is that highlighting key words in text misses the context, flow, and relationships contained. In addition, the words used in sentences and paragraphs are seldom in a format suitable for building a cross-referenced index of ideas. Why?

The idea-normal format

Index entries should reflect a classification scheme — noun, descriptor — noun, verb — verb, object. Outside of topic and section headers, most ideas in text are in a linguistic rather than a classification format.

The synonym problem

So, for these reasons, the MARK program may not be the best method of capturing the ideas contained in the text. That's why even the most sophisticated relational-text database finds (on the average) only 22 percent of the possible cases to cite in legal research.

Using HOUDINI

Language control

Sometimes, I use HOUDINI instead of MARK to build my language maps of the information in my ASCII files. HOUDINI provides two advantages over MARK — language control and file management. Here's what I mean:

In using HOUDINI, I load in my list of filenames (output file from the FS splitter program). Then I make input links to each file, describing the ideas contained in the file. After linking each file to my description of the ideas contained, I sort the descriptions to find the files in common with each idea. Why is this better?

Back to taxonomies

One big reason! First, I keep better control over my vocabulary, meaning I always use the same language to describe the same idea regardless of where or when I find it. This is important because with most hypertext, you don't just index information. Instead, you create taxonomies (logical classification schemes) to organize information which is a magnitude beyond indexing.

Whorf said language determines reality, meaning that seeing and understanding is limited to working vocabulary.

If you want others to understand previously unknown information, they first must acquire the classification patterns.

Pause for a minute. All indexes presume you know the taxonomies of the subject, but almost no index shows the actual taxonomy. However, if you're building hypertext that transmits knowledge, you need to first create then put portions of your classification system of ideas into every decision point in both your network and ASCII links. If you do this, users can't help but acquire the organizing patterns (i.e., knowledge) of the subject. For example, a basic physiology text book always contains an index to the words contained. But the index never contains the taxonomy (classification system of the words) describing the nine basic systems of the human body.

If you don't know the taxonomy, you can't effectively use the vocabulary to find what you want. However, if you can see the taxonomy, you can quickly use the system and acquire the vocabulary.

Structuralism

That type of thinking is called structuralism — and explains why you can't acquire knowledge from relational databases. They find language and display information, but ignore structure. This also explains why not everyone can create hypertext. Creating hypertext requires talents for abstracting and generalizing a level beyond running conventional application software.

While everyone can use hypertext, not everyone can create hypertext systems.

That leads to how I use HOUDINI. After input-linking my description of the ideas contained in each file to a file, I classify my descriptions. Continuing this process, I build my taxonomy and keep my language usage consistent. That's one of my important secrets. Here's the other.

Contemporary semantics

Here's a 17th-century concept of finding the meaning in text.

Linguistic concepts of "meaning"

Words are like beads on a string . . . and to understand the meaning of a necklace of beads (sentences or paragraph), you only have to understand the roots (etymology, oldest meaning) of each of the words. I think this is garbage. Yet most computer text search programs are based on this type of thinking.

Semantic concepts of "meaning"

Now, consider a contemporary semantic approach to finding ideas and meaning.

Shallow and deep structures

Text is a "shallow structure" as it uses conventions of form, grammar, and language to describe what was in the authors mind ("deep structure"). Finding the meaning the author intended requires thinking about the deep structures implied from analysis of the shallow structures. How do you do this?

The theory suggests that you look beyond word and phrase translation to larger units (single and multiple paragraphs). You try to identify the contained objects, events, actions, abstractions, and relationships that the author uses in these larger units.

I use this (decomposition of text into classes of information) to help me identify and organize ideas. For example:

How to identify ideas

Types of Ideas	Identification of the idea type
Objects	Nouns — you can touch objects
Events	Nouns — events have time and location
Actions	Verbs — actions change something
Abstractions	Nouns — you can't touch abstractions
Relationships	Connectivity between objects, events, etc.

Vocabulary control

Using this guide to finding ideas, I write my descriptions of any idea contained in the text to include what kind of idea it is. This makes easy work of keeping my vocabulary consistent which aids the organizing of all ideas (building a classification scheme).

Sounds great! But here's the problem. I like to use two computers — one containing HOUDINI with my file and language network, the other displaying each file in turn.

Using your word processor

For another approach to the indexing of ideas, I sometimes use only my word processor to index the ideas in each file. Here's how:

I set up two windows — one for the current file; one for my cumulative list of ideas in each file.

Then, I use macros to identify the beginning of a new file, to append marked text from my word processor window to my idea index, and to automatically transfer my comment lines from the text file to the idea file.

When finished, I use MaxThink to convert my word processor list into my idea description, filename, and file description format for sorting by SREF.

Why use a word processor?

Why do I use the word processor? If I'm consistent with my descriptions (always using the recommended format), I don't need to develop my taxonomy as I go along.

What should you do? I'd recommend that you try all three ways of building your language lists (ideas <filename> file description) in order to enter your ASCII cross-links. As I see it:

Three approaches to identifying cross-links

The MARK program is the most systematic approach.

The HOUDINI program provides the best control for taxonomy-building.

Your word processor is the most flexible.

Pick the one that matches your needs and understanding of the process.

Checking links

Chapter 11

HyperLink includes several ways of validating the links you make. Let's look at each.

REFALL program records all the hypertext jumps in all the files in a directory. Here's how it works:

The output format of REFALL is a two-level MaxThink outline. The first level lists each of the <filenames>; the second level or subtopic list lists the contained <filename> jumps (one per line).

Run REFALL in the same directory as the files because it also checks each file in each subdirectory

The REFALL program extracts the angle-bracket filename information from ASCII, MaxThink, or HOUDINI files, but not from network files. For that reason, include the source files (MaxThink or HOUDINI) used to create your networks in the same directory as the ASCII files.

To run the program on the files in the FILES directory in drive C:, do the following:

Action	**Results**
Type: REFALL C:\files LIST	Records all the files and jumps in the files directory to the file LIST

I use this LIST file created by REFALL in a number of different ways to validate my networks — by itself, with the IC program, with MaxThink, and with HOUDINI. Let's look at each.

Reading the REFALL output

I'll scan the printed output from REFALL looking for errors in the listed hypertext jumps in each file. Generally, I don't want to see too many (seven or more) or too few (two or less).

How much is too much?

When files have too many jumps, they should either: (1) be split into multiple files, or (2) be put into a network file. Why?

Each ASCII file should provide a focused explanation and include jumps to expanded information on particular points contained in the file. In contrast, network files should provide clear either/or choices between ways of gathering information.

Don't clutter a file with hypertext jumps to the same explanatory file.

Continuing, I also scan the REFALL listings to see if files contain duplicate jumps to the same topic. I'd rather include one instead of many jumps to the same topic in a file. Why?

Don't cry "Wolf" or say "important" too often

I'd like each jump to reward the user with new information instead of repeating familiar information. Why? Because users cannot predict what they'll find behind each jump. If the information is new and useful, it encourages exploration. But if most jumps only rehash the familiar, then the user becomes biased against hypertext jumping, which negates the purpose of hypertext.

For that reason, each file should not contain too many jumps to or from it. REFALL identifies files with too many exit jumps. In a moment, I'll show you how to use HOUDINI to find files that have too many inputs.

Using REFALL with the IC program

Don't jump to missing files

Here's another class of errors I want to avoid. I don't want <filename> links to non-existent files. How can this happen?

Don't jump to misspelled filenames

If I misspell a filename in the hypertext jump or forget to include the referenced file in my system, I've accidentally made a dead-end link. Here's how I check my systems for dead-end links.

The IC program (Integrity Checker) uses the file created by the REFALL program as an input. The IC program then verifies that each of the files exist in the system for each of the claimed <filename> jumps.

If the file isn't found, the program either prints or writes out the error listing. The error listing shows the file containing the jump (with the file in <filename> format). The error <filename> jumps are listed as subtopics (MaxThink format).

I use this output to verify that my system is complete. However, there's a more complex issue behind the IC program. What are the legal <filename> jumps?

As I write this, I'm also modifying PC-Hypertext to handle the following types of <filename> jumps:

Hints of what's coming

<phone xxx-xxx-xxxx>	Dials the phone number xxx
<TIFF filename>	Display the TIFF format file
<POST filename>	Show Display Postscript (EPS)

As a result, the text in angle-brackets will contain parameters beyond the filename itself. This means the IC program will shortly change to test for errors in these new classes of jumps.

How do you stay current? If you phone me, I'll keep you posted of the changes and provide updates to PC-Hypertext and the programs in HyperLink.

Using the REFALL file in my word processor

In many ways, the REFALL listing of all the files and all their contained jumps would be more useful if each <filename> contained a following description. That information is found in the file created by the FS (SPLITTER) program. How do you attach the file descriptions to the REFALL structure of all the ASCII links?

In building systems of 500 files, there are 500 minutes in a long day. I automatically convert modification minutes per file into days of effort.

While tedious, I load my word processor with the output of the SPLITTER program in one window, and the output of the REFALL program in the other. I block mark the description in the SPLITTER window, then copy it after each occurrence of the specific filename in the REFALL program.

That alone is one reason why most people will never build hypertext systems of any consequence.

This process may take 30 minutes with 100 files and 300 cross-links. Once finished, you'll have an invaluable map on paper of your entire ASCII-file links. I use this map as a guide for making modifications and improvements to the files contained, or as an input to constructing my networks. More on this in a minute.

Using the REFALL file in MaxThink

As the REFALL output file loads directly into MaxThink, you might think that MaxThink can add significant insights to the information contained. Wrong! Here's why.

The ASCII-file structure of links without descriptions of each of the files doesn't offer much opportunity for MaxThink. However, if you add descriptions to each of the links, then MaxThink is very important. Here's why.

First add the descriptions, then load the information into MaxThink.

I first build links between my ASCII files, then build my network links. With the REFALL information which shows the links in my ASCII files, I can use MaxThink to build my network links.

Building links by classifying

Recipe for using MaxThink

Use the BINSORT command of MaxThink to categorize all the ASCII files. The hierarchies you create become the access paths to each file. In order to provide suitable access paths to a specific file, you need to classify your files in as many ways as possible.

Merge each of these different hierarchies into a common file, then use the M2H program to translate this file into your hypertext network file.

Using the REFALL file in HOUDINI

Saving the best way for last

HOUDINI provides the most interesting way to use the REFALL file. As HOUDINI can read in MaxThink files and merge duplicate filenames, the REFALL file in HOUDINI becomes a full network. Here's why that's important:

If you F5 mark a topic in HOUDINI and use the FILE OUTPUT command, HOUDINI partitions your knowledge. Wow, what does that mean?

The FILE OUTPUT command traces all the output links from one file to other files, the output links from the other files to more others, and the output links from the more other, and so on.

Set closure in knowledge

Now, "set closure." HOUDINI finds the files that form the set closure of any files. This is important because each closed set of files forms a "knowledge area." Once you've identified a knowledge area, you can build a separate network that separates the files in just that area. You can then link each of your knowledge areas into a master network. Slick? You bet!

Number of references to a file

I mentioned that each ASCII-file should not have too many input or output links. I described how to use the REFALL list to track the number of output links from each file, but I didn't describe how to track the number of input links to each file.

How to see the number of references to a particular hypertext node

If you load the REFALL file into HOUDINI, you can easily trace the number of files that reference each specific file by using the LEFT LINK menu or the Ctrl-Left command. Good!

More to come!

Together, the methods mentioned help you verify the links you've constructed. Both MaxThink and HOUDINI can use the REFALL information to build your network links. Read the next chapter on the PERM program which helps build decision trees before you use the network construction methods mentioned in this chapter.

Building network links

Chapter 12

I've already mentioned several methods for building networks. This chapter covers the details of using MaxThink, HOUDINI, or the PERM program to make networks of links to access each ASCII file. Let's look at each.

Using MaxThink to build networks

You could use MaxThink and its BINSORT command to classify each of the files in your hypertext system. Normally, you'd classify your information from many different viewpoints to serve the needs of various users, merge the resulting outlines, then use M2H to convert the file into the PC-Hypertext format.

Because files can contain more than one idea, the number of indexed ideas in a hypertext system grows faster than the number of files.

While this may work with 20-40 files, it is impossible with systems of hundreds or thousands of files. Why? Because the number of possible knowledge sets (classification schemes or hierarchies) grows at a faster rate than the number of files.

Difficult? Yes, because MaxThink doesn't provide a way of identifying the various possible classification schemes. That's why I recommend HOUDINI for network building. Read on!

Using HOUDINI to build networks

If you used HOUDINI to index the ideas in each of the files, you've almost got your network built. Here's what needs to be added.

Should you build a separate taxonomy for each group of users?

If you categorize the ideas contained in your network (i.e., organize a taxonomy) into several hierarchies (to serve multiple needs or viewpoints), you've got your network. Look at it this way.

The purpose of your ASCII links is to provide expanded information on particular points. The purpose of your network links is different. Your network links should provide a choice of paths such that the user can easily see where each might lead, as well as identify the desired choice. Why? Two reasons.

Generally, the minimum number of keystrokes is 10 or less

First, you want the user to reach the desired information within a minimum number of keystrokes. That only occurs if the user can predict what is beyond each choice.

Second, you want the user to acquire an understanding of the overall structure of the knowledge system without necessarily traversing all paths.

Create predictable paths

If a user can predict where a choice might lead (even if not explored), the structure of that information is understood and remembered. As a consequence, over time, the user of such information networks acquires a good understanding of all the information in the systems.

Differences between MaxThink and HOUDINI

Just a quick note. MaxThink classifies information using one dimension per level. HOUDINI classifies information using from 5-100 dimensions per level. This is the fundamental difference between organizing information using hierarchy and using network manipulators.

Using the PERM (decision tree builder) program

Suppose I wanted to build a decision tree program that, through a minimum set of questions, would separate dime-store balls. For example, the balls have mixtures of the following properties:

Price	39¢, 49¢, 59¢, not known, two conflicting prices
Color	red, green, blue, red and blue, color not known
Size	small, medium, large
Bounce	none, some, lots, not known
Texture	soft, smooth, hard, fuzzy, soft and fuzzy

Now with this five-dimension problem, let's also say that some people think that price is most important. Others are more interested in other attributes. How do you build the network? Hint: It's easy with PERM. Let's see how.

You need to use the PERM program to understand this application of resolution theory.

The PERM program collects this information from three MaxThink files to produce a fourth hierarchical file. Load this fourth file into HOUDINI (with the merge duplicates turned OFF), save the HOUDINI file, then run H2H to produce your network. First, the three MaxThink files.

The first MaxThink file created is the ATTRUBUTE file. Make an outline containing at the first level BALL-A, BALL-B, . . . , BALL-N, etc. As subtopics of each item include the specific information about price, color, size, bounce, and texture. Save this as file A.

The second MaxThink file created is the CATEGORY file. Modify the first outline (LEVELIZE with FENCES, then DELETE) so that you have a list containing all the attributes. Classify the attributes so that the first level contains category topics such as price, color, texture, size, and bounce; the subtopics contain the associated attributes. Edit the category topic (price) to create questions (i.e., What is the price, What is the color?). Save this file as file B.

The third MaxThink file created is the PRIORITY file. List all the possible users of the information (i.e., child, mother, shopkeeper, friend, baseball player, etc.). As subtopics, list the questions (What is the price?, etc) that each would like answered in the order of importance to each. Save this as file C.

Congratulations! Now, you want to produce the file OUTPUT that contains the minimum set hierarchy of questions. In order to separate balls, for each market segment, do the following:

Action
Type: PERM A B C OUTPUT

Results
Produces the file OUTPUT

Replace the balls with filenames and replace items with short descriptions of what's in a file. Using a minimum set of questions, PERM automatically produces decision trees that separate one file from another. Interesting!

If you are clever, you can use PERM to build all your transitive links (minimum paths to files). However, you'll still need to use the techniques mentioned in the previous chapters to build your explanatory links (cross-links between files).

What are the limits?

PERM is limited to 256 files, attributes, categories, and markets which means that it can build optimum hierarchies of 256 levels deep.

In practice, a 10-level, four-choice hypertext system does separate four to the 10th power ideas. (This may be more than the number of independent ideas in the universe.) Oh!

Chapter 13

I've already mentioned changes that I'll shortly make to PC-Hypertext. The hypetext features I'm working on are:

What's coming:

<phone xxx-xxx-xxxx>	Hypertext jump to dial phone
<TIFF filename>	Display TIFF graphics
<PostScript filename>	Display PostScript graphics
<NAPLIPS filename>	Display NAPLIPS graphics
<CD-ROM files>	Display CD-ROM information
<Laser disk positions>	Display Laser disk information
Automatic similarity maps	Visual display of relationships
Automatic flow diagrams	Easiest path through information

In a few weeks, these changes will be available to you at no cost.

In distributing PC-Hypertext with several large (two megabyte) systems with monthly 500K additions, I've tried several systems — straight floppy disks, compressed files formats, multi-volume compressed formats, and auto install routines.

Should you compress your distribution files?

While the available compression schemes offer (on the average) a 50 percent reduction of disk space, I've concluded that straight floppy disks (360K format) are the easiest to produce and easiest for users to install. I'd recommend the same unless your users are experienced in un-compressing programs.

Protecting the users against accidental changes

One of the consequences of personal computing is that no security systems can prevent people determined to change a system. However, most users would like some assurance that their knowledge system is working as intended.

I use TOUCH to date-stamp the files associated with each of my hypertext updates.

My first approach (TOUCH program) set the file creation date and time of all the files in a directory to a uniform value, then flagged any changed files whenever accessed by PC-Hypertext. However, in distributing knowledge systems via a series of updates, this system seemed inappropriate.

The DS program records the standard.

The CF program lets the users check their system against the standard.

My current approach is to include with each update a file listing the size, date, and name of every file in the entire system. The program DS (Disk Status) creates this file, which is called CHKLIST.HYP. Along with the CHKLIST.HYP file, I include the program CF (Check Files).

Any user of my knowledge systems can run the CF program. If any of the files have a date, time, or size different from those on the master system, the program lists the files needed to return the knowledge system to its intended format. This approach has proved sufficient to protect against accidental changes.

Send me a copy of your hypertext efforts for a copy of the AN program

My other security system, the ENCODE program, has one unmentioned feature. I have a program called AN (Add Name) that I use to quickly insert the name of each user into the top line of the main menu of PC-Hypertext. Having each user's name highly visible in the knowledge systems I build works well in my markets.

Just for review, the ENCODE program produces an encrypted file (necessary to run PC-Hypertext). If changed in any way, the program won't boot. The AN (Add Name) program uses the same encryption scheme to include the user's name in the FILELIST.HYP file.

Conclusion

In many ways, the computer is still at the tinkering stage . . . requiring expertise similar to the early days of the auto, radio, phone, or airplane.

For example, the current relational approaches to information resemble tinkering because they expect users to master the language, synonyms, set operations, and combination searches needed to find information. Few people want tinkering approaches to knowledge.

Push-button Information

We live in a push-button society. Obtaining knowledge should be as simple as dialing a phone . . . it should not require users to build a phone each time they wish to access a system of knowledge.

Simply stated, if you can build systems that provide users with whatever knowledge they want just by pushing the arrow keys, your systems will replace all other methods of dispensing information, knowledge, or expertise.

You probably know the start-up stories of the current giant firms selling software for processing information. Even better start-up opportunities exist for today's hypertext developers and publishers who can organize knowledge. Why?

Hypertext construction is a solo process (like writing novels) instead of a factory process (like linking code) than offers income opportunities unmatched elsewhere in computing. Call for details!

Appendix

The first two pages of Appendix display two different methods for organizing the utilities. These methods are:

Alphabetical by name	— this page
Classification by function	— next page

The classification listing of the utilities on the next page describes the sequence, purpose, and use of each utility.

AN	add name or text to encrypted FILELIST.HYP file
ANGLE	makes hypertext links to files in a directory
BIGSORT	organizes hypertext links into categories
CF	identifies changes in a hypertext system
CONNECT	shows generations of links to/from files
DS	records status of files in a hypertext system
ENCODE	creates FILELIST.HYP file
FS	makes ASCII files into hypertext nodes
H2A	converts HOUDINI files to ASCII hypertext
H2H	converts HOUDINI files to .NET format
HYPER	PC-Hypertext program (uses .HYP/.NET files)
IC	checks integrity of all links
INVERT	shows all hypertext jumps to a file
M2A	converts MaxThink files to ASCII hypertext
M2H	converts MaxThink files to .NET files
MARK	makes hypertext jumps to files by key ideas
MATCH	makes hypertext links to defined phrases
MP	makes hypertext links to words in a directory
REFALL	shows all hypertext jumps from a file
RMP	makes hypertext links to non-garbage words
REPAIR	automatically corrects error in links
SREF	collects ideas into hypertext categories
TK	translate keystrokes from keylog file
TOUCH	sets all file date stamps to a specified value
WF	determines word frequency of words in a directory

At the DOS prompt, type each filename for a description of the program and the paramters for running it.

HyperLink Utilities — Run each program to display its description

Make hypertext nodes	FS	makes ASCII files into hypertext nodes
	M2A	converts MaxThink files to ASCII file hypertext
	H2A	converts HOUDINI files to ASCII file hypertext
Create hypertext links	ANGLE	makes hypertext links to all files in a directory
	MATCH	makes hypertext links to defined phrases
	MARK	makes hypertext jumps to files by key ideas
	MP	makes hypertext links to words in a directory
	RMP	makes hypertext links to non-garbage words
	WF	determines word frequency of words in a directory
Organize hypertext links	BIGSORT	organizes hypertext links into common categories
	SREF	collects ideas into hypertext categories
Manage hypertext links	IC	checks integrity of all links
	REFALL	shows all hypertext jumps from a file
	REPAIR	automatically correct error in links
	CONNECT	shows generations of links with specified files
	INVERT	shows all hypertext jumps to a file
Monitor hypertext system	CF	identifies unwanted changes in a hypertext system
	DS	records status of all files in a hypertext system
	TOUCH	sets all file date stamps to a specified value
Essentail files for running a hypertext system	HYPER	PC-Hypertext program. Support files are: TITLE.HYP, VERSION.HYP, HELP.HYP, CONFIG.HYP, FILELIST.HYP
Programs to create encryption files, .NET files, or parce keystorke logs	AN	allows modification of FILELIST.HYP file
	ENCODE	creates encrypted FILELIST.HYP file
	H2H	converts HOUDINI files into .NET files
	M2H	converts MaxThink files into .NET files
	TK	converts keystroke files into audit trail file

AN — add name to FILELIST.HYP title line

DOS Description

AN allows users to change the title line at the top of the PC-Hypertext screen for example to insert a desired user name, serial number or date.

The ENCODE utility creates a FILELIST.HYP file. This encrypted file contains the desired title line text. To modify this line of text (contained in the FILELIST.HYP file), do the following at the C:\HU prompt

Action	**Results**
Insert disk into drive A:	Preparation to modify file on A:
Type: AN	Loads AN program from C:\HU

The AN program displays the current title line and provides a line editor to modify the line.

The commands of the line editor are:

Right arrow	Move cursor to the next character
Left arrow	Move cursor to previous character
Ctrl-Right arrow	Move to beginning of next word
Ctrl-Left arrow	Move to beginning of previous word
Home	Move to beginning of line
End	Move to end of line
Ins key	Toggle between insert/overtype

Related Utilities:
HyperLink includes seven utilities to handle issues of security of knowledge systems that you create — AN, ENCODE, H2H, M2H, DS, CF, and TOUCH.

AN allows you to put a users name into a distribution disk.
ENCODE lets you control the network files accessed by each disk.
H2H and M2H use binary formats to prevent data modification.
DS and CF identify accidental changes to a knowledge system.
TOUCH uses common dates to identify accidental changes.

ANGLE — makes hypertext links to all files in a directory

DOS Description:

```
ANGLE -- Generates Angle Bracketed filenames for a directory
(C) MaxThink 1989  44 Rincon, Kensington, CA  94707
input          Directory name
output         Angle Bracketed Filenames
usage:  angle  input_directory_name  output_name
```

Example To produce a file xxx containing hypertext links to all files in directory c:\hr, type:

Angle c:\hr xxx

Usage ANGLE helps me rapidly analyze or summarize the contents of a large number of files.

(1) Run ANGLE to create hypertext jumps to each file
(2) Load HR
(3) Load TR-WORD with the file created by ANGLE
(4) Call HR to make hypertext jump on first displayed link
(5) Press ESC to enter your description after the hypertext jump
(6) Repeat steps (4) and (5) for each link on the list

While I like using ANGLE then TR and HR together, here's a different way to accomplish this same abstraction task:

(4) Press F4 to hypertext jump to file on the cursor line
(5) Press Esc to return to the editor to enter your description

Finally, with hypertext jumps to long complex files, do the following:

(5) Press Alt-E to make a window containing the hypertext file
(6) After Zooming window, explore usingnext window command
(7) When finished, press F3 ESC to close window, then repeat (5)

Related utilities
I use ANGLE primarily to create input files for TransText

Limitations
Check on sub categories

BIGSORT — organizes hypertext links into common categories

DOS Description

```
BIGSORT -- Sorts the file
(C)  MaxThink 1989  44 Rincon, Kensington, CA, 94707
Input          File to be sorted
Output         Sorted file
usage: bsort input_filename output_filename
```

Example To sort the word file WORDS in c:\hr and write the results to file xxx in the current directory, type the following:

Bsort c:\hr\words xxx

Usage I sort ASCII files to organize either filename, words, or hypertext links, depending on which is listed first on each line. If I need to change the order of the fields, I run a TR-WORD macro on the file to flip the fields before sorting.

To sort small ASCII files that are under 60K in size, I use the DOS SORT command as follows: SORT <xxx >yyy — sorts xxx to file yyy

However, because the MP (multiple phrase) and RMP (Reverse Multiple Phrase) easily create ASCII files far larger than 60K in size, I use BIGSORT.

Related utilities:
As BIGSORT runs as fast as SORT, I've stopped using SORT because it crashes the computer if you accidentally try to sort a file larger than 60K.

Limitations:
In operation, BIGSORT reads the input file 60K at a time, writes the results to a temporary file and repeats the process to the end of the input file. BIGSORT picks the next item from among the temporary files and writes it to the output file.

In this process, the limitation is disk space that is equal to twice the size of the input file.

CF — identifies unwanted changes in a hypertext system

DOS Description

```
CF — Checks the system filelist
(C) MaxThink 1987   44 Rincon, Kensington, CA  94707
FILE CHKLIST.HYP must exist to run CF
```

Example To check if all the files in the current directory and all subdirectories match (in number, size, and date/time) now match those at the time the DS program created the CHKLIST.HYP file, then list the changes to file xxx, do the following:

```
CF > xxx
```

Usage After creating a hypertext system (perhaps containing thousands of ASCII), I run the DS program to create a file call CHECKLIST.HYP which records the name, size, date, and time of all files in all directories.

After installation, the CF command reports on the security of the system. The CF program rapidly compares the current system against that contained in the CHKLIST.HYP file. Any deviation from the delivered system (in files, location, size, or time) are listed to the screen.

To check all files in the specified directory and associated subdirectory, after running the DS program, use your word processor delete the files from the CHKLIST.HYP file that you do not wish to check. This feature is useful on a LAN to create a system that notifies you only of changes to files that you are concerned about.

Related utilities:
Use the DS (Disk Status) program to create the CHECKLIST.HYP file.

Limitations:
None

CONNECT — shows generations of links into or out of specified files

DOS Description

```
CONNECT — Generates Files Connectivity
(C) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input    Uses file generated by REFALL
Input    Contains the filenames to be connected
Output   Reports the connectivity
usage: connect file1 file2 file3
```

Example To create a file showing all generations of input and output links (i.e. file A links to B, file B links to C, etc) to a group of files listed in file yyy, having already run the REFALL program to create file xxx, then write the result to file zzz, do the following:

CONNECT xxx yyy zzz

Usage I use CONNECT to analyze hypertext networks for partitions. To define the idea of a partition, consider this example:

(1) File A has links to 5 other files
(2) Each of these files have 5 links to the next generation
(3) Repeat this process of tracing all multiple-generation links
(4) Results — some systems trace to all files — no partition exits
(5) Results — some systems trace to few files forming a partition

Set of files that form a partition (link only among themselves) represent a module of knowledge. Such modules are highly desired because:

(1) Hypertext systems containing partitions are easier to expand and update

(2) Because partition files are cross-linked, the only addition hypertext effort is adding a taxonomy (classification) of links to the files in the partition.

CONNECT — Related utilities:
Run the REFALL, IC, and REPAIR programs first to insure that your hypertext links contain no errors.

Use ANGLE and TransText to create a list of the input list of files for exploring connectivity.

The output file from CONNECT program first lists all files that through multiple connections link into files on the specified list, then lists all the file that can be reached from at least one file on the specified list.

For a trick, I use the macros in TransText to create a batch file that runs the CONNECT program on each file in my entire system. This rapidly identifies the important files (reached from many files), the connective files (reach many files), and possible partitions (reach few files).

Limitations:
None

DS — records status of all files in a hypertext system

DOS Description

```
DS — Recursively generates filenames under each subdirectory
(C) MaxThink 1987   44 Rincon, Kensington, CA
Input          Directory name
Output         CHKLIST.HYP used by CF program
usage: ds directory_name
```

Example To create a file (CHKLIST.HYP) in the current directory that records the current status (name, number, date, size) of all files in a specified directory and associated subdirectories, type the following:

DS C:\HR

Usage After creating a hypertext system (perhaps containing thousands of ASCII), I run the DS program to create a file call CHECKLIST.HYP which records the name, size, date, and time of all files in all directories.

After installation, the CF command reports on the security of the system. The CF program rapidly compares the current system against that contained in the CHKLIST.HYP file. Any deviation from the delivered system (in files, location, size, or time) are list to the screen.

However, the DS and CF programs are very useful in multiple-user LAN systems. Use them to keep track of change in files in your directory (the comment files in teamware environments) or changes to the entire system.

Alternately, if specific files in a network are important to you, run DS and edit the CHKLIST.HYP file so that the CF program checks for changes only in those files. This allows you to "camp" on files with automatic notification of changes by others. Clever!

Related utilities:
The DS program creates files used by CF program.

Limitations: None

ENCODE — encrypts the title and list of .NET files

DOS Requirements

```
ENCODE — Creates FILELIST.HYP file from ASCII file
(c) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input    ASCII file listing PC-Hypertext system/network files
Output   FILELIST.HYP file (default if not otherwise specified)
```

Example Consider this example ASCII file named TEST:

```
Demo Hypertext System — First example
HYPER.NET
FILES.NET
LIST.NET
```

The first line prints at the top line of PC-Hypertext screen
The second line lists the first network file loaded
Third to last lines list other network files for loading

To create the FILELIST.HYP file needed to run a hypertext system using the network files HYPER.NET, FILES.NET, and LIST.NET, do the following:

ENCODE TEST

Usage The ENCODE program allows developers to destribute hypertext systems containing encrypeted FILELIST.HYP files that selectively control access to portions of the entire system.

Related utilities:
Use the HOUDINI or MaxThink programs to create a network files.

Use the H2H and M2H utilities convert the HOUDINI and MaxThink files into a binary network format required by PC-Hypertext.

Use the AN (Add Name) program to insert a users name into the title line contained in the FILELIST.HYP file.

Limitations: None

FS — makes ASCII files into hypertext nodes

DOS Description

```
FS — Creates up to 128 files from single ASCII file using code
below (C) MaxThink 1987  44 Rincon, Kensington, CA  94707
Make new file     ==== <filename>  description
Skip text         ==== skip
Append text       ==== write
usage: fs input_filename output_filename
```

Example To split a marked file xxx (has ==== <filename> description entered at split points) into separate files (each with the assigned filename) and write a separate summary file yyy listing each file (as a hypertext jump) along with its description, do the following:

FS xxx yyy

Usage While I could use the Block Write commands in TransText to split a large ASCII file into a series of smaller files, I prefer FS because FS:

(1) is fast (with familiar text, I can create 100 nodes per hour)
(2) creates an output list, showing jump and file descriptions

If I use TransText to enter the split codes, I also format the headers of subsequent files as well as add sequence-links. For sequence links, I add links to the next and previous files. At the time I enter the split codes, I add and format my header text — title, source, subject, date — after each split mark. To standardize the format, I create a macro in TransText that inserts a standard format, then modify the format with the editor.

Related utilities:
My original vision of the hypertext construction process was:

(1) FS to create hypertext nodes
(2) MARK to identify ideas in each hypertext node
(3) SREF to find ideas that occur in multiple files

While I've added utility programs for steps (2) and (3), FS remains one of my general-use programs.

Limitations: FS is limited to creating 128 files per run.

H2A — converts HOUDINI files to ASCII file hypertext networks

DOS Description

```
H2A — Transfer Houdini file to ASCII files
(C) MaxThink 1988  44 Rincon, Kensington, CA  94707
Houdini filename
Output   Starting ASCII filename (AlphaNumeric only)
usage: h2a  input_name  output_base_name
```

Example To convert a file xxx created by the HOUDINI network builder into a series of ASCII files with the first starting at file name LINK01 (subsequent are LINK01, LINK02, etc) with each file containing hypertext jumps that match the related output links, type the following:

H2A xxx LINK01

Usage While the five different hypertext systems — in MaxThink, HOUDINI, PC-Hypertext, TransText, and HyperRez — I've created each have particular advantages, experienced users of all my software have concluded:

(1) HOUDINI is a powerful tool for cross-linked networks
(2) TransText is a powerful editor for ASCII hypertext
(3) H2A coordinates the power of HOUDINI and TransText

For users of HOUDINI, H2A does the following:

(1) Removes all limits found HOUDINI (2500 topics /7500 links)
(2) Hypertext graphics using PC-Hypertext and PC-Paintbrush
(3) Provides more flexible screen layouts

For non-users of HOUDINI, take a look at the demo version of HOUDINI that includes a two-hour cassette tutorial ($10) after you built several hypertext systems.

Related utilities:
H2A links HOUDINI to the TransText and HyperRez programs

Limitations:
Make the HOUDINI files rooted meaning that a HOUDINI file (1) has a hierarchical order in the output links, and (2) that the first file in the system is the root topic in the hierarchy.

H2H — converts HOUDINI file to .NET format

DOS description

```
H2H — Converts Houdini files to PC-Hypertext format
(c) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input     Single or wildcard filename (do not use .NET extension)
Output    Adds .NET extension to input filenames
usage:    h2h input_file_list
```

Example To read a HOUDINI file named XXX then create a network file named XXX.NET, do the following:

H2H XXX

Usage The network file screen in PC-Hypertext appears similar to that of the RIGHT LINK display in HOUDINI. For that reason, I build and polish my networks in HOUDINI then convert them using H2H.

Because the GOTO command in PC-Hypertext can execute programs, I create a .BAT file that loads HOUDINI with my network file, erase the old .NET file, and runs the H2H. Then to change any network relationship in PC-Hypertext, I press G to call the GOTO command and enter this .BAT file.

For background, the HOUDINI files are ASCII while the .NET files are binary. This binary format loads faster and also prevents others from making undesired changes to your networks.

Related Utilities:

M2H converts MaxThink files to the .NET format

Limitations: If a previous .NET file already exists, the program does not create a new .NET file. Why? This utility, like many of the others I've created, will run in a batch mode. Making it interactive prevents such unsupervised batch operation.

IC — checks integrity of all links

DOS Description

```
IC — Checks integrity of REFALL file
(C) MaxThink 1987   44 Rincon, Kensington, CA  94707
Input          Uses file generated by REFALL
Output         Specify file or listing of reference errors
usage: ic  input_filename out_filename
```

Example To check for errors in a system of files containing hypertext jumps, after running the REFALL program to create file xxx, and have the errors (links to non-existent files) recorded in file yyy, do the following:

IC xxx yyy

Usage In size, my largest hypertext systems have contained 3,000-5,000 files and more than 50,000 links. Yet regardless of size, IC is a most important program. Non-functional links don't belong in any hypertext system. IC rapidly identifies these errors.

Related utilities:

(1) Run REFALL to create the input file necessary for IC.
(2) Run REPAIR to rapidly correct errors identified by IC.

Limitations: None

INVERT — shows all hypertext jumps to a file

DOS Description

```
INVERT — Inverts REFALL output
(C) MaxThink 1988  44 Rincon, Kensington, CA  94707
Input    Uses file generated by REFALL
Output   Inverted REFALL output
usage: invert input_file output_file
```

Example The produce a file yyy showing for each file all the files that make hypertext jumps into it, having already run the REFALL program to create a file xxx, do the following:

INVERT xxx yyy

Usage INVERT helps me analyze my hypertext system.

From a classical hypertext viewpoint, the most important topics are the classification topics that organize information in ways that allow users to rapidly find what they want. The REFALL program shows these link relationships.

From a different viewpoint, the most important topics are the definition topics that clearly define the boundaries and limits of ideas. Definition topics typically have many input links which are easily shown by the INVERT program.

Related utilities:
Examining all hypertext links from a file is easy — I either read the file (if it is short) or look at the REFALL file (eliminates all the text).

Examining all hypertext links to a file is also easy — I either use INVERT (shows immediate links) or CONNECT (shows all generations of links).

Limitations: None

M2A — converts MaxThink files to ASCII file hypertext networks

DOS Description

```
M2A — Transfer MaxThink file to ASCII files
(C) MaxThink 1988  44 Rincon, Kensington, CA  94707
Input    MaxThink filename
Output   Starting ASCII filename (AlphaNumeric only)
usage: m2a input_name output_base_name
```

Example To convert a file xxx created by the MaxThink Idea Processor into a series of ASCII files with the first starting at file name LINK01 (subsequent are LINK01, LINK02, etc) with each file containing hypertext jumps that match the related output links, type the following:

M2A xxx LINK01

Usage While the five different hypertext systems — in MaxThink, HOUDINI, PC-Hypertext, TR-WORD, and HyperRez — I've created each have particular advantages, users of all my software report:

(1) MaxThink rapidly builds good systems of classification
(2) TransText extends the hypertext capabilities of MaxThink
(3) H2A coordinates the power of HOUDINI and TransText

For users of MaxThink, M2A does the following:

(1) Removes limits found in MaxThink (4000 topics /12000 lines)
(2) Provides hyper-graphics using PC-Hypertext/PC-Paintbrush
(3) Provides more flexible screen layouts

For non-users of MaxThink, look at my demo version of MaxThink that includes a two-hour cassette tutorial ($10) after you have built several hypertext systems.

Related utilities:
(1) MaxThink creates hierarchies.
(2) M2A converts MaxThink files into ASCII hypertext systems.
(3) TransText edits ASCII hypertext system.
(4) PC-Hypertext and HyperRez run finished hypertext system.

Limitations: Limited by MaxThnk — even handles clone topics!

M2H — converts MaxThink file to .NET format

DOS description

```
M2H — Converts MaxThink files to PC-Hypertext format
(c) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input    Single or wildcard filename (do not use .NET extension)
Output   Adds .NET extension to input filenames
usage:   m2h input_file_list
```

Example To read a MaxThink file named XXX then create a network file named XXX.NET, do the following:

H2H XXX

Usage The network file screen in PC-Hypertext appears similar to that of the RIGHT LINK display in HOUDINI. However, I often build and polish my networks in MaxThink because the editor is more powerful than the one in HOUDINI.

To make the screen layout of MaxThink match that of the right box in PC-Hypertext, change the left margins to match and the right margin to 32 characters. For background, the MaxThink files are ASCII while the .NET files are binary. This binary format loads faster and also prevents others from making undesired changes to your networks.

Related Utilities:

H2H converts HOUDINI files to the .NET format

Limitations: If a previous .NET file already exists, the program does not create a new .NET file. Why? This utility, like many of the others I've created, will run in a batch mode. Making it interactive prevents such unsupervised batch operation.

MARK — makes hypertext jumps to files by key ideas

DOS Description

Screen display of MARK with a file containing text.

```
Now is the time for all good people to think

F1Help  F2Word  F5Mark  F7Load  F8Quit
(C) MaxThink 1987 (415) 428-0104
```

Screen display after pressing F1 key in MARK

```
CURSOR MOVEMENT                    CURSOR JUMPS
To previous letter  Left arrow     To top of screen Home Home
To next character Right arrow      To end of screen End End
To previous word  Ctrl-Left arrow  To previous page  PgUp
To next word      Ctrl-Right arrow To next page  PgDn
To end of word Ctrl-End            File start Home Home Home
To end of file  End End End

To start of a line      Home
To end of a line        End        Other command
To next line            Down arrow F1 (toggle)
Load new file           F7 xxx     Quit program  F8

Marker (highlight) command
Mark words              F5 (toggle)
Add marked words        Enter
Unmark words            Esc
Add synonyms            F2
```

Guide to running MARK program

To mark a word or phrase:
(1) At DOS, type MARK to load the program
(2) Press F7 then enter the file name
(3) Use arrow keys to move the cursor to beginning of a word
(4) Press F5 to turn on highlight
(5) Press Ctrl-Right to move cursor to end of word
(6) Press Enter to add marked word
(7) To add a synonym, press F2 then enter the synonym
(8) When finished, press F7 to load a new file

Related utilities:
ANGLE, MP, RMP, and MATCH also create hypertext links.

Limitations: Needs HELP.MRK file for F1helps.

MATCH — makes hypertext links to defined phrases

DOS Description

```
MATCH — Find text patterns in directory (and subs)
(C) MaxThink 1987  44 Rincon, Kensington, CA  94707
 usage: match A B C D
A: -iinput_header_filename (optional)
B: -ooutput_filename (default to screen)
C: -ddirectory_name (directory name for search)
D: text patterns ...(prompts if not entered)
hit Escape Key to abort
```

Example To make a file xxx listing hypertext jumps (and the relevant line of text) in all files in directory c:\hr containing any one of up to ten user-prompted text patterns, do the following:

Match xxx c:\hr

Usage I originally designed MATCH as a program that was called from within PC-Hypertext. It worked as follows:

(1) File xxx contained text describing the use and call to MATCH
(2) A hypertext jump to file yyy showed the text of file xxx
(3) User selected the call to MATCH in file yyy
(4) MATCH program prompted for text to search for
(5) MATCH program created file yyy after adding file xxx header
(6) User returned to a new file yyy with the result of the search

While this sounds complex, this application of the MATCH program let users with the PC-Hypertext program build lists of hypertext jumps to desired words simply by making a hypertext jump. That's why the program includes parameters for input file headers.

To see an example, download the HAZ-CHEM.ARC from my BBS (phone 415-540-6114).

Related utilities:
TransText contains five different utilities that automatically make links:

ANGLE	Format . . . <file>
MATCH	Format . . . <file screen line> plus the line of text
MP	Format . . . phrase followed by <file screen line>
RMP	Format . . . phrase followed by <file screen line>
MARK	Makes . . . phrase followed by <file>

Here's the tasks that these programs solve:

Task	**Programs**
Find text occurrences in a single file	TransText FIND
Find text occurrences in multiple files	MP
Find text occurrences in all files	MP or MATCH
Convert text occurrences to ideas	MP, BIGSORT
Find key word phrases	WF, RMP
Link ideas to files	MARK, SREF

I use the above search programs to find relevant words (which is linguistics), examine the words in context then capture the idea contained in the words using definition, synonyms, and boundaries (which is semantics).

Words alone are not ideas. However, by looking at the same words in various contexts (using hypertext jumps), you can easily abstract their meaning from their usage.

Communicating meaning is central to hypertext. Hypertext should not be jumps to words, but jumps among the classifications of ideas. However, each classification may include jumps to the words in context perhaps to demonstrate the scope or nuances contained in a idea.

Limitations:
The MATCH program can build files that are larger that the capacity of HyperRez (20K file size limit) or TransText (400K file size limit).

MP — makes hypertext links to defined phrases in a directory

DOS Description

```
MP — Match Phrases for files under each directory
(C) MaxThink 1988  44 Rincon, Kensington, CA  94707
Input    Phrase filename
Output   Matched filename
Input    Filenames (?, *, and directory_name are acceptable)
usage: mp input_phrase_name output_name filenames
```

Example To search specified files (or directory) c:\hr*.* to make hypertext jumps to the specific line of all occurrences of specified text phrases listed in file xxx, then write the results to file yyy, do the following:

MP c:\hr*.* yyy zzz

Usage For text searches that construct hypertext links, MP is my most useful program for the following reasons:

(1) The input file can contain up to 1024 phrases to search for
(2) The wildcard filenames search one to a directory of files
(3) The output format (phrase - hypertext jump) sorts to concepts

Before running MP, I often run WF (word frequency) to find all the words contained in a directory of files. This list helps me create a list of words to search for. Alternately, if the source text contains a good index, I use the index as a guide to create my phrase file.

In practice, I limit my phrases to the various possible expressions of a single idea to shorten the run time of the MP program. To search for multiple ideas, I create a batch file using MP to write the hypertext jumps to each idea to separate files.

Related utilities:
ANGLE, MARK, MATCH, and RMP also create hypertext links

Limitations: Maximum number of search phrases is 256.

REFALL — shows all hypertext jumps from a file

DOS Description

```
REFALL — Shows angle bracket references as subtopics of each
filename (C) MaxThink 1987   44 Rincon, Kensington, CA  94707
Input          Directory name containing files (ASCII .NET, etc.)
Output         Single file (MaxThink format) for IC program
usage:   refall directory_name output_filename
```

Example To search all files in directory xxx (and files in the associated subdirectories) for hypertext jumps creating a file yyy that lists each file and its contained hypertext jumps, do the following:

REFALL c:\hr yyy

Usage REFALL is my abstracting program that converts a complex network of hypertext ASCII files into a logical system of hypertext jumps from each file. I use the program as follows:

(1) I load the REFALL file into HOUDINI to study the loops, chains, sub-sets, and supersets within the link structure.

(2) REFALL shows me the link density (number of links in each file). I reduce the number of links in files with too many links (or split the file), and I question the inclusion of files to few links.

(3) I run IC (Integrity Checker) on the output file from REFALL to identify dead-end links (links to files not in the system) or mis-typed links (links to files already in the system if typed correctly).

(4) To locate the most important (or least important) files in a system, I run INVERT on the output file from REFALL. INVERT creates a file showing all the files that directly link into each file.

(5) To locate partitions (sets of files that link only to each other), I run CONNECT on the output file from REFALL. CONNECT shows all generations of files that eventually link to or that can be reached from specified files. Files with small numbers of such links indicate a possible partition of knowledge. Files with a large number of such links possibly identify the connective files that unite multiple units of knowledge.

Related utilities:
The IC, INVERT, and CONNECT programs use the output file created by REFALL.

Limitations: None

REPAIR — automatically correct errors in links

DOS Description

```
REPAIR — Fixes Errors Generated by IC Program
(C) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input    Uses file generated by IC
usage: repair input_filename
```

Example After running REFALL (lists all hypertext jumps), IC (lists faulty links), and editing the IC file xxx to enter corrections, do the following to automatically correct the link error in each of the relevant files:

REPAIR xxx

Usage One way to correct link errors uses TransText to open, find and correct, then close each relevant file containing a "bad" link.

Here's an easier way to correct the obvious errors in filenames found in the output file from IC. Load the error file into TR-WORD, add the correction after each incorrect filename, run REPAIR, then IC again.

For format, it is your option to include or not include the angle brackets with the corrections.

For removing links in files, enter the angle brackets with no contained text (i.e., <>).

To change the name of a file that is referenced by many other hypertext files, run INVERT on the output file from REFALL. Next, erase all files form the output of INVERT except for those that connect to the specific file, add the new filename after each reference, run REPAIR, then change the filename itself.

Related utilities:
I use the REFALL, IC, and INVERT programs with the REPAIR program.

Limitations: If the replacement filename in the hypertext jump differs in length from the initial filename, the REPAIR program shifts the location of text to the right of the change.

RMP — makes hypertext links to non-garbage word clusters

DOS Description

RMP — Reverse Match Phrases for files under each directory (C) MaxThink 1988 44 Rincon, Kensington, CA 94707	
Input	Garbage filename
Output	Matched filename
Input	Filenames (?, *, and directory_name are acceptable)
usage: mp input_garbage_name output_name filenames	

Example After creating file xxx that lists the words to suppress, to make a file yyy that contains hypertext jumps to the non-garbage words found in files link0* in directory c:\hr, do the following:

RMP xxx yyy c:\hr\link0*

Usage I use the RMP program as follows:

(1) I Run WF (word frequency program) to identify all the words in the relevant file(s) or directory

(2) I create a file containing the words to eliminate

(3) I run RMP which identifies each sentence, eliminates the "garbage" words in each sentence, then makes links to each of the remaining phrases

(4) I load the resulting output file into TransText to rapidly delete unwanted text or modify the key phrases

(5) Alternately, I run BIGSORT on the resulting file to sort the phrases, then edit them convert from the original words (linguistic) to ideas with meaning and boundary (semantic)

Because the parameters for RMP (and most of the other utilities) are entered after the program name (prompt entry not used), I can run the RMP program from a .BAT file for analyzing many different files from different viewpoints.

I have also used the RMP program (with only "xxx" in the suppression file) to create a file that separates each sentence with a hypertext jump to that sentence. Clever!

Related utilities:
The BIGSORT and MP (locates specified phrases) programs support RMP in the following ways:

BIGSORT sorts the output file (sometime five times larger than the source);

MP is the inverse of RMP. MP makes links to specified words while RMP suppresses links to specified words.

Limitations:
Number of words in the "garbage" list is limited to 256.

SREF — collects ideas into hypertext categories

DOS Description

```
SREF — Assembles files from MARK program into single sorted
file (C) MaxThink 1987  44 Rincon, Kensington, CA  94707
Input          Use file wildcard as output files of MARK program
Output         Single file (do not use same filename extension)
usage: sref output_filename input_filename_list
the extension part of an input filename must be '.REF' the
extension part of the output filename may not be '.REF'
```

Example After running the MARK program (which creates .REF files for each marked file), to combine each of the .REF file into a single file name xxx that alphabetically lists all words with their associated hypertext jumps to their file, do the following:

SREF link*.REF xxx (if MARK used files link1, link2, . .)

Usage The MARK which I might use to identify key words in 200 files, would also create 200 files. Each of the created files is named identically to its source file except for the file spec (three character extension), which is .REF

The SREF collects all the words contained in all 200 .REF files into a single sorted file with each word or phrase containing a hypertext link to its source file.

Once collected, I load the file into TransText, then execute each of the jumps, editing or adding modifiers to each word or phrase to clarify the meaning of each. This process helps me develop a controlled vocabulary (single expression for each idea) for the referenced material.

Once my vocabulary is uniform, a quick scan of the SREF file (Sorted and combined REF files) shows the following.

Two or three reoccurring words suggest perhaps a circular hypertext connection between the words in each file. This would show each in context.

Five to eight reoccurring words suggest hypertext jumps from each to a single file. The single file explains the concept (definition file) and contains links showing the context of each word. This is a "start" form of hypertext.

Ten or more reoccurring words suggests a hypertext system that shows the subclasses of the ideas behind the word, again with jumps between the subclasses as well as to each word in context.

Related utilities:
SREF is similar to first joining a large number of files, then using BIGSORT to organize the words.

Limitations: None

TK — translates keystrokes from keylog files

DOS description

```
TK — Transform PC-Hypertext keylog file
(c) MaxThink 1987 44 Rincon, Kensington, CA 94707
Input       Uses file generate by PC-Hypertext keylog file
Output      Transformed PC-Hypertext keylog file
usage:      tk  input_filename  output_filename
```

To convert file XXX created by the <LOGON XXX> command of PC-Hypertext to file YYY, do the following:

TK XXX YYY

Usage: PC-Hypertext includes three unusual angle-bracket commands:

<LOGON xxx>	When selected, it closes any previous log file then records all subsequent keystokes to file xxx.
<LOGOFF>	When selected, it closes the current log file.
<FIELD description>	Calls a one line editor for entry of text after the description. This information is included in the log file.

The log file records in sequences each user action in hypertext jump format. The TK program converts the codes of commands to words such as DOWN, RIGHT, UP, etc.

In usage, once you have the location in the ASCII or network file along with the subsequent keystrok, you can create as simple program to automatically analyze and report on the path each user selected through the network.

This feature is useful for analyzing information access, testing user abilities or understanding, or automatically creating reports of usage.

Related utilities: None

Limitations: None

TOUCH — sets all file date stamps to a specified value

DOS Description

```
TOUCH — Sets all file dates/times to single value
(C) MaxThink 1987  44 Rincon, Kensington, CA  94707 usage:
TOUCH [-fFILENAME] input_filelist (may contain wildcard)
-f will use FILENAME's date/time, otherwise system's date/time
```

Example To set the date/time stamp on files LINK* to the date/time of file xxx, do the following:

TOUCH -fxxx LINK*

Usage In systems of large numbers of files, I often use touch to set all the files to a single date/time stamp. At a later time, this quickly identifies the files that have been changed (by me or others).

In using TransText in a "teamware" setting, at the end of each session, use TOUCH to set the files in your directory to a uniform date/time. That makes it easy to identify which of your comment files contains new information (different time) whenever your begin a new session.

Related utilities:
The DS and CF programs can be used in place of TOUCH to identify changes in a system of hypertext files. The advantage of DS and CF is that they identify changes across multiple subdirectories.

Limitations: TOUCH.COM must be in same directory as the files that it changes.

WF — determines word frequency of all words in a directory

DOS Description

```
WF — Word Frequency Statistics
(C) MaxThink 1987  44 Rincon, Kensington, CA  94707
Parameter     [-C]: Case Sensitive  [-N]: No Count
Input         Filenames (?, *, and directory_name are acceptable)
usage:        wf [-C] [-N] filelist
```

Example To write a sorted list of all word in file c:\HR\LINK01 to file xxx, do the following:

WF C:\HR\LINK01 > xxx (redirects output to file xxx)

Usage I use WF with the count on to look at the alphabetical distribution of words in a file, groups of files, or a directory.

For a list of words organized by frequency, I run BIGSORT of the WF output list.

To enter words created by WF on a "garbage" for RMP, I run WP with the -N option to remove the word count information, then erase all but the desired "garbage" words.

Related utilities:
I use WF with the BIGSORT, MP, and RMP programs.

Limitations: None

Index

AN utility 49
ANGLE utility 50
ASCII files, how to cross link 31
ASCII files, adding links to 21
ASCII hypertext nodes, creating 25
audio-visual features 12
automatic sub-directories 27

batch files, building networks with 20
BINSORT utility 51
CF program, how to use 45
CHECK FILES program 45
CHKLIST.HYP file, description of 45
classification, importance of 4
collecting knowledge 22
CONFIG.HYP 15
CONNECT utility 53
construction software 3
copyrights and hypertext 24
critical reading 27
CTRL.COM 15

decision trees, how to build 43
deep structures 36
dictionary programs 7
directories, recommended setup of 16
disk files, PC-Hypertext 15
disk files, HyperLink 6
disk files, converting 18
DISK STATUS program 45
DS program, how to use 45
DS utility 55

ENCODE program, purpose of 46
ENCODE program, how to use 19
ENCODE utility 56
explanation hypertext links 31

file directories, recommended 16
FILELIST.HYP, description 14
FILELIST.HYP, how to create 19
FILELIST.HYP, using AN to modify 46
FILENAME.NET network file 14
files of HyperLink 6
files of PC-Hypertext 15
files, converting to network 18
FS program, details about the 26
FS utility 57

grammar checking programs 7
graphic browsing 40

H2A utility 58
H2H program, using the 18
H2H utility 59
help screens to PC-Hypertext 10
HELP.HYP 14
HOUDINI files, how to convert 18
HOUDINI, building networks with 42
HOUDINI, compared to MaxThink 43
HOUDINI, importance of 4
HOUDINI, indexing information 41
HOUDINI, indexing ideas with 36
HYPER.EXE 14
hypertext, construction software 3
hypertext, copyright issues 24
hypertext, definition of 1
hypertext, filename conventions 27
hypertext, how to make nodes 26
hypertext, opportunities 46
HyperLink, overview 5
hypertext, paper linking 33
HyperLink, software not included in 46
hypertext, types of construction 22
hypertext, value of 2
hypertext, word processing and 6
hypertext jumps, types of 10
hypertext links, types of 31
hypertext links, number of 32
hypertext systems, validation of 38

IC (Integrity Check) program, using 42
IC utility 60
ideas and words in text 37
ideas, classification of 37
index construction 37
instructions, PC-Hypertext 9
INVERT utility 61

jumps, type of hypertext 10

keystroke logs 12
King, Stephen 8
knowledge collection 22
knowledge, classification of 37
knowledge, formats of 23
knowledge, public domain 24

links, types of 31

M2A utility 62
M2H program, using the 19
M2H utility 63
MARK program, how to use the 33
MARK utility 64
MATCH utility 65
MaxThink files, how to convert 18
MaxThink, compared to HOUDINI 43
MaxThink, indexing ideas with 40
MaxThink, use of 4
MESSAGES.HYP 14
MP utility 67

network links, how to build 42
networks, creating 18
networks, rooted 18

optical character recognition (OCR) 23

paper linking 33
PC-Hypertext, future releases of 45
PC-Hypertext, help screens for 10
PC-Hypertext, information in 11
PC-Hypertext, keystroke log of 12
PC-Hypertext, manual 9
PC-Hypertext, operations 8
PC-Hypertext, required files 14
PC-Hypertext, special features 12
PC-Hypertext, updates to 13
PC-Hypertext, updates to 40
PC-Hypertext, use of 3
PC-Hypertext, word processor and 12
PERM program, how to use 43
philosophy, MaxThink 4
public domain information 24

REF 15
REF file, how to create the 35
REFALL program, listing from 39
REFALL program, how to use 38
REFALL utility 68
REPAIR utility 70
RMP utility 71

semantics, theories of 36
shallow structures 36
software used for hypertext 3
Splitter program, details of the 26
SREF program, how to use the 35
SREF utility 73
star hypertext 32
structuralism, ideas of 36
structures, types of 36
structure, importance of 1
sub-directories, automatic 27
sub-directory calls 12
synonym inclusion 34

table of contents ii
taxonomy, building a 36
taxonomy, importance of 2
thesaurus program 7
TITLE.HYP 14
TK utility 75
TOUCH program, how to use 46
TOUCH utility 76

updates to PC-Hypertext 13

validating links 38
vocabulary control 35

WF utility 77
word processing and hypertext 29
word processor linking 12
work processor, adding links with a 21
word processor, hypertext 6
word processor, structure with a 37

MaxThink -- Software to extend your mind . . . in ideas, creativity, and imagination

Knowledge Processing

Software for developers:

TOOLS

MaxThink
44 Rincon Rd.
Kensington, CA
94707
Phone (415) 428-0104